НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Веб-застосунок для оренди та продажу нерухомості»

Виконав:
студент IV курсу, групи ІА-52
Земляной Олексій Олегович _____

Керівник:

Ст. вик. Моргаль О.М. _____

Рецензент:

_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 рік

АНОТАЦІЯ

Земляной О.О. Веб-застосунок для оренди та продажу нерухомості. КПІ ім. Ігоря Сікорського, Київ, 2019.

Проект містить 62 с. тексту, 35 рисунків, посилання на 30 літературних джерел, додатки та 4 конструкторських документа.

Ключові слова: веб-застосунок, оренда та продаж нерухомості, C#, ASP.NET Core, Entity Framework Core, база даних.

Об’єктом розробки є веб-застосунок для оренди та продажу нерухомості. Мета розробки – створення веб-застосунку для розміщення оголошень про оренду та продаж нерухомості.

Дипломна робота присвячена розробці веб-застосунку для оренди та продажу нерухомості з використанням мови програмування C#. Результатом роботи став додаток, розроблений за допомогою технологій ASP.NET Core та Entity Framework Core. Були досліджені також такі підходи, як ASP.NET, ASP.NET MVC та ASP.NET Web API, та технологія для доступу до баз даних ADO.NET. Було виділено основні недоліки та переваги вищезазначених технологій в розробці додатків. Також було розглянуто основні патерни програмування та розробки, правила та рекомендації щодо побудови архітектури таких додатків.

В результаті було спроектовано та розроблено серверну частину веб-застосунку для оренди та продажу нерухомості.

SUMMARY

Zemlianoi O.O. Web application for renting and selling real estate property. Igor Sikorsky KPI, Kyiv, 2019.

The project contains 62 pages, 35 figures, links to 30 literary sources, annexes and 4 design documents.

Keywords: Web application, renting and selling real estate property, C#, ASP.NET Core, Entity Framework Core, database.

The object of development is creating a web application for renting and selling real estate property. The purpose of the development is to serve ads for renting and selling real estate property.

This work is devoted to the development of a Web application for renting and selling real estate property using the C# programming language. The result was an application developed by ASP.NET Core and Entity Framework Core. Also, approaches such as ASP.NET, ASP.NET MVC and the ASP.NET Web API, and technology for accessing databases ADO.NET were also explored. The main drawbacks and advantages of the above-mentioned technologies in the development of applications were highlighted. Also discussed were the main patterns of programming and development, rules and guidelines for building the architecture of such applications.

As a result, the server part of the web application for renting and selling real estate property was designed and developed.

ЗМІСТ

ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ

ООП – об’єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних;

DI – Dependency Injection;

EF – Entity Framework;

HTTP – HyperText Transfer Protocol;

IoC – Invetrion of Control;

SQL – Structured Query Language;

SSMS – SQL Server Management Studio.

ВСТУП

У сучасному світі все поступово йде до глобальної автоматизації та комп’ютеризації. Сьогодні дуже важко знайти компанію, яка не користується IT технологіями для розвитку власного бізнесу. Оскільки найбільше для цих цілей використовують Інтернет, то зовсім не дивно, що саме web-додатки здобувають все більшу популярність.

Актуальність роботи полягає в тому, що дуже важко знайти продукти що тебе цікавлять саме як споживача. Так, наприклад, в аналогічних додатках для розміщення оголошень з продажу чи оренди квартири не відбувається відповідного контролю власника. Потенційному покупцеві не доведеться спеціально йти до агенства нерухомості чи агентства, що займається пошуком орендодавців під конкретні вимоги. Все це можна буде здійснити не виходячи з дому, а саме – через Інтернет.

Метою роботи є розробка web-додатку з використанням сучасних технологій програмування та розроблення бази даних, яка буде стійкою до змін.

Завданням роботи є розробка програмного забезпечення, а саме його серверної частини, яке буде надавати можливість користувачам в зручній для них формі вибирати певні оголошення залежно від власних потреб.

1. ПОНЯТТЯ ВЕБ-ЗАСТОСУНКУ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття веб-застосунку

Під час розгляду літератури в конкретній області слід зазначити декілька важливих нюансів. По-перше, саме створення додатку. По-друге, його оновлення. По-третє – обслуговування. Також дуже важливо зазначити, що спроектована система є WebApi[1] спроетована на мові програмування C#[2].

При розгляді літератури в області дослідження інформаційних систем слід підкреслити декілька важливих аспектів, таких як: створення, оновлення та обслуговування такого web додатку.

Що стосується створення такого продукту в першу чергу треба дати поняттю «server», отже з точки зору «заліза», «web-сервер» - це комп'ютер, який зберігає файли сайту (HTML-документи, CSS-стилі, JavaScript-файли, картинки та інші) і доставляє їх на пристрій кінцевого користувача (web-браузер і т.д.). Він підключений до мережі Інтернет і може бути доступний через доменне ім'я, подібне mozilla.org[3].

З точки зору програмного забезпечення, web-сервер включає в себе кілька компонентів, які контролюють доступ web-користувачів до розміщених на сервері файлів, як мінімум - це HTTP-сервер. HTTP-сервер - це частина програмного забезпечення, яка розуміє URL (веб-адреси) і HTTP (протокол, який браузер використовує для перегляду web-сторінок)[4].

Перед тим як ми зрозуміємо що таке WebApi, давайте подивимося що таке API(інтерфейс прикладного програмування).

У комп’ютерному програмуванні інтерфейс прикладного програмування (API) являє собою певну сукупність підпрограм, протоколів та інструментів для створення програмного забезпечення та додатків.

Простіше кажучи, API – це свого роду інтерфейс, який має набір функцій, що дозволяють програмістам отримувати доступ до певних функцій чи даних додатку, операційної системи або інших служб.

WebApi, виходячи з назви, - це API інтерфейс в Інтернеті, доступ до якого можна отримати по протоколу HTTP. Це концепція, а не технологія. WebApi можна створювати з використання різноманітних технологій, таких як Java, .NET і т.д.

Веб-API ASP.NET[5] являє собою розширюване середовище для створення служб на основі HTTP, до яких можна звертатися з різних додатків різних платформ, таких як веб, Windows, мобільні пристрої чи інше[6].

Серверний Web API - це програмний інтерфейс, що складається з однієї або декількох відкритих кінцевих точок до певної системи повідомлень про відповідь на запит, як правило, виражається в JSON або XML, що відкривається через веб-інтерфейс. Mashup - це веб-додатки, які поєднують використання декількох веб-API на стороні сервера. Webhooks - це веб-інтерфейси API на сервері, які приймають за вхідні дані уніфікований ідентифікатор ресурсу (URI), який призначений для використання як віддаленого іменований пайплайну або зворотного виклику, так що сервер виступає в ролі клієнта для розмежування наданого URI і ініціювання події на іншому сервері, який обробляє цю подію.

Кінцеві точки є важливими аспектами взаємодії з серверними Web API, оскільки вони вказують, де лежать ресурси, доступ до яких може отримати програмне забезпечення третіх сторін. Зазвичай доступ здійснюється через URI, до якого публікуються HTTP-запити, і звідки очікується відповідь.

Кінцеві точки повинні бути статичними, інакше неможливо гарантувати правильне функціонування програмного забезпечення, яке взаємодіє з ними. Якщо місце розташування ресурсу змінюється (а разом з ним і кінцева точка), то попередньо написане програмне забезпечення порушиться, оскільки необхідний ресурс більше не може бути знайдений на тому ж місці. Оскільки постачальники API все ще хочуть оновлювати свої Web API, багато з них ввели систему ідентифікації версій в URI, що вказує на кінцеву точку. Якщо вирішити оновитись до другої версії, вони можна це зробити, зберігаючи підтримку для стороннього програмного забезпечення, яке використовує першу версію.

Web API на стороні клієнта - це програмний інтерфейс для розширення функціональності у веб-браузері або іншому HTTP-клієнті. Спочатку вони були найчастіше у вигляді нативних розширень браузера, однак більшість нових цільових одиниць націлена на стандартизовані прив'язки JavaScript.

Так як система клієнт-серверна, в якості серверу виступає IIS Express[7]. Це легка автономна версія IIS оптимізована для розробників. Вона дозволяє використовувати останню версію IIS для розробки і тестування веб-сайтів. IIS Express має всі можливості IIS 7 і вище, також додаткові функції, що були розроблені для спрощення розробки веб-сайтів, включаючи:

– він не працює як служба і не потребує прав адміністратора для виконання більшості задач;

– IIS Express гарно працює з додатками ASP.NET і PHP;

– декілька користувачів IIS Express можуть працювати незалежно на одному комп’ютері.

Зазвичай для створення подібної системи спочатку проектують базу даних. Оскільки найпоширенішою і найпопулярнішою мовою для проектування баз даних є SQL, саме її і вибрано[8].

Що стосується систем управління базами даних, то слід зауважити,що таких систем існує досить багато, таких як: PostgreSQL, Oracle, SQL Server Management Studio (SSMS), MySQL[9].

SSMS – це інтегроване середовище для управління будь якою інфраструктурою SQL, від SQL Server до SQL Azure[10]. SSMS являє собою інструмент для налаштування, моніторинга та адміністрування баз даних. Її можна використовувати для запитів, проектування і управління репозиторіями даних, де б вони не знаходилися – на локальному комп’ютер чи в хмарному сервісі. Безперечною перевагою є те, що вона безкоштовна.

Для роботи з базою даних використовується фреймворк Entity Framework(EF) Core[11]. Це легковісна, розширювана і кросплатформенна технологія з відкритим кодом.

EF являє собою спеціальну об’єктно-орієнтовану технологію на базі фреймворка .NET для роботи з даними. Якщо традиційні властивості ADO.NET дозволяють створювати підключення, команди та інші об’єкти для взаємодії з базами даних, то EF являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з базою незалежно від типу сховища[12]. Якщо на фізичному рівні ми оперуємо таблицями, індексами, зовнішніми ключами, то на концептуальному рівні, який нам пропоную EF, ми вже працюємо з об’єктами.

Центральною концепцією EF є розуміння сутності або ж entity. Сутність являє собою сукупність даних, що асоціюються з певним об’єктом. Саме тому дана технологія передбачає роботу не з таблицями, а з об’єктами та їх наборами.

Відмінною рисою EF є використання запитів LINQ для вибірки даних з БД[13]. За допомогою LINQ ми можемо не тільки витягувати певні рядки, об’єкти, що зберігаються, з БД, але і отримувати об’єти пов’язані різними зв’язками.

Систему розроблено з урахування основних принципів SOLID[14]. Це акронім, введений на початку 2000-х, який позначає п’ять основних принципів об’єктно-орієнтованого програмування і проектування.

У наш час усі інформаційні системи проектуються за допомогою високорівневої мови програмування. Найпоширенішим є технологія .NET Core, яка представляє широку ініціативу для реалізації реактивної підтримки на всіх рівнях стек-веб-розробки, безпеки, даних, обміну повідомленнями тощо[15].

Під час створення програмних систем використання принципів SOLID допомагає створити таку систему, яка буде легка в підтримці і розширенні впродовж довгого часу. Принципи SOLID – це керівництва, які так само можуть бути використані під час роботи з вже існуючим проектом для його покращення та вдосконалення. Розглянемо кожен з принципів:

– The Single Responsibility Principle – принцип єдиної відповідальності. Кожен клас виконую лише одну задачу;

– The Open Closed Principle – принцип відкритості/закритості. Програмні сутності повинні бути відкритими для розширення, проте закриті для модифікації;

– The Liskov Substitution Principle – принцип підстановки Барбари Лісков. Об’єкти у програмі повинні бути замінені на екземпляри їх підтипів без змінення правильності роботи програми;

– The Interface Segregation Principle – принцип розділення інтерфейсів. Багато інтерфейсів розроблених для різних клієнтів, краще за один інтерфейс для всіх клієнтів;

– The Dependency Inversion Principle – принцип інверсії залежностей. Об’єкти повинні залежати від абстракцій, а не від конкретної реалізації.

Також під час написання коду багато уваги приділялося C# Code Conventions[18]. Це встановлені правила з того як потрібно писати код. Вони слугують декільком цілям:

– вони створюють узгоджений вид коду для того щоб читач міг зосередитися на логіці, а не на макеті;

– вони дозволяють читачам швидше зрозуміти код, даючи припущення, що засновані на попередньому досвіді;

– вони полегшують копіювання, змінення та обслуговування коду;

– вони демонструють найкращі практики C#.

Дуже велику стратегічну роль у пректі відіграє Dependency Injection(DI)[16].

DI або ж впровадження залежностей це механызм, який допомагаю сладувати певним правилам проектування системи для того щоб реалізувати слабкі зв’язки між об’єктами системи. Найчастіше це відбувається через впровадження абстракцій та інтерфейсів. Це в свою чергу робить систему більш гнучною та адаптивною.

Зазвичай для впровадження залежностей використовують IoC-контейнеры (Inversion of Control). IoC - це принцип програмного забезпечення, за допомогою

якого контроль об’єктів або частин програми переноситься в контейнер або структуру. Це найчастіше використовується в контексті об’єктно-орієнтованого програмування[17].

Переваги цієї архітектури:

— відокремлення виконання завдання від його реалізації;

— що полегшує перемикання між різними реалізаціями;

— більша модульність програми;

— більша зручність в тестуванні програми, виділяючи компонент або знущаючись з її залежностей, і дозволяючи компонентам спілкуватися через контракти.

Розглянемо приклади існуючих рішень веб-застосунків для оренди та продажу нерухомості.

1.2 Веб-застосунок LUN

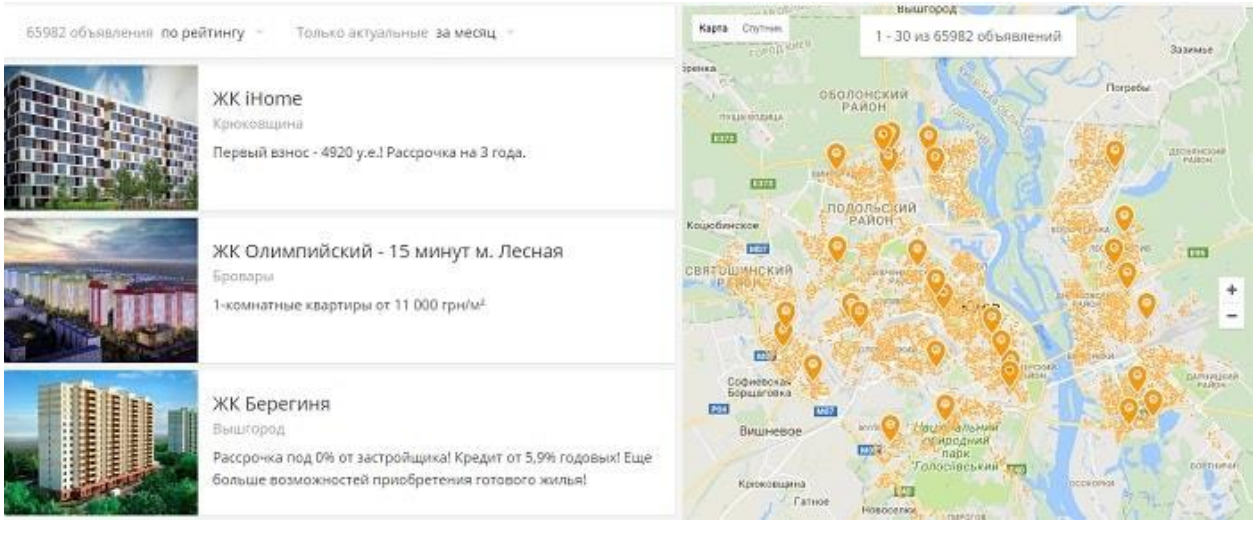

Рисунок 1.1 – інтерфейс веб-застосунку LUN

Весь сайт розбитий на дві частини, одна з яких нерухома, друга - функціональна. Сайт спеціалізується на продажу і оренді нерухомості первинного і вторинного ринку. Графа пошуку передбачає поступову і багаторівневу фільтрацію. На головній сторінці можна вибрати бажану дію щодо нерухомості і місто. Далі виконується перехід на наступну сторінку, де користувачеві запропоновані відбіркові фільтри, виходячи з яких внизу відкриваються пропозиції відповідно до запровадженим запитам. Він зручний, практичний, а його лаконічність в дизайні дозволяє зосередитися на своїй цілі, не відволікаючись на інше.

1.3 Веб-застосунок EST!

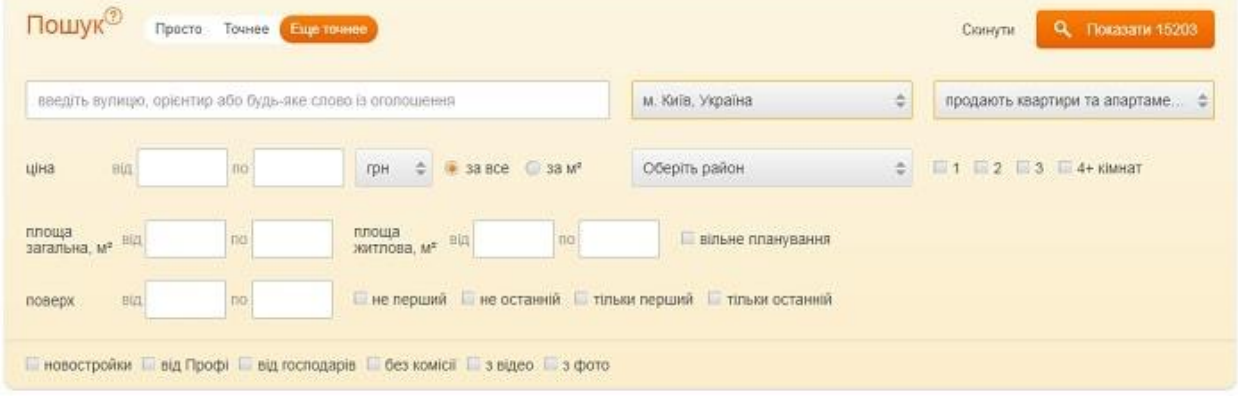

Рисунок 1.2 – інтерфейс веб-застосунку EST!

Цей приклад сайту з купівлі та оренді нерухомості і його продажу. Тут відвідувач може знайти будь-яку інформацію не тільки про об'єкт, але і ринку нерухомості в цілому. Є розділ статей, блогу, клубу закордонної нерухомості. Під шапкою з спливаючих меню розмістилася графа пошуку. І тут користувачеві запропоновано три варіанти: просто, точніше, ще точніше. У міру переходу від категорії до категорії відкривається більш складна система фільтрів. Нижче на головній сторінці викладені гарячі пропозиції з фото і цінами. З лівого боку навігація для переходу в окремі категорії. Внизу окремим списком розбиті

конкретні пропозиції. Відвідувач може вибрати окрему категорію, якщо його цікавлять, наприклад, тільки котеджі або пансіонати.

1.4 Висновки до розділу 1

У даному розділі було розглянуто основні поняття веб-застосунків. Також описано найпопулярніші веб-застосунки України для оренди та продажу нерухомості.

Як можна бачити, вже існують відповідні веб застосунки з різноманітним функціоналом, що необхідний клієнту для розміщення своїх повідомлень або пошуку інших. Проте, відсутня реалізація, де були б присутні одночасно можливості сиворення власного профілю користувача, можливість створювати оголошення як для оренди, так і для продажу, пошук по ключовим словам.

В даній роботі буде розроблено серверну частину веб-застосунку, що містить в собі вищезазначений функціонал.

2. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ

В даному розділі буде розглянуто сучасні технології, які можна застосувати для створення соціальної мережі на мові C#.

Спочатку буде розглянуто основні можливості технології ASP.NET, буде описано, що саме лежить в основі ASP.NET.

Далі будуть більш детально описані такі підходи, як ASP.NET MVC, ASP.NET Web API та ASP.NET Core. Найцікавішим є, звичайно, ASP.NET Core, адже цей фреймворк являє собою повний перепис, який об'єднує раніше окремі ASP.NET MVC та ASP.NET Web API у єдину програмувальну модель.

Може виникнути таке питання, навіщо взагалі використовувати C#, адже можна писати програми на Java чи на C++ та Swift, що є досить перспективними та цікавими мовами програмування. Дійсно, але мова C# має багату історію, цікаві конструкції, фреймворки, велику спільноту. Тому хотілося б написати основні та найбільш цікаві можливості даної мови:

1. Інкапсульовані сигнатури методів, звані делегатами, які підтримують типобезпечні повідомлення про події.

2. Властивості(properties), що виступають в ролі методів доступу для закритих змінних-членів.

3. Атрибути з декларативними метаданими про типи під час виконання.

4. Вбудовані коментарі XML-документації.

5. LINQ, що пропонує вбудовані можливості запитів в різних джерелах даних.

6. Підтримка асинхронних операцій.

7. Використання TPL – Task Parallel Library в .NET.

8. Використання широкого спектру класів .NET та фреймворків.

Як видно, дана мова має багато гарних можливостей, вона схожа на C++ та легша в засвоєнні.

2.1Технологія ASP.NET

ASP.NET — технологія створення веб-застосунків і веб-сервісів, яка була створена компанією Microsoft. Ця технологія є основною частиною платформи Microsoft.NET і розвитком старішої технології Microsoft ASP. На цей час останньою версією цієї технології є ASP.NET Core 2.0.

ASP.NET дуже схожа на стару версію ASP зовні, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішній устрій ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі .NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Після випуску сервера Internet Information Services 4.0 в 1997 році, компанія Microsoft почала досліджувати можливість нової моделі веб-застосунків, яка задовольнить скарги на ASP, особливо пов'язані з відділенням оформлення від змісту, і яка дозволить писати «чистий» код. Робота з розробки такої моделі була доручена Марку Андерсу, менеджеру команди IIS, і Скотту Гутрі, що прийшов на роботу в Microsoft в 1997. Андерс і Гутрі розробили початковий проект протягом двох місяців, і Гутрі написав код первісного прототипу під час різдвяних канікул 1997 року.

Початковий проект називався «XSP»; Гутрі пояснив в інтерв'ю 2007 року що, «завжди запитують, що означає буква X. У той час вона нічого не значила. XML починається з неї; XSLT починається з неї. Все кльове починається з X, тому ми його так і назвали.» Прототип XSP був написаний на Java, але скоро було вирішено побудувати нову платформу на основі Common Language Runtime (CLR), бо на платформу Java у компанії Microsoft закінчувалась ліцензія. Гутрі

описав це рішення як «величезний ризик», тому що успіх нової розробки був пов'язаний з успіхом CLR, яка, як і XSP, перебувала на ранній стадії розробки.

Корпорація Майкрософт рекомендує використовувати в динамічному коді програми code-behind model, яка розміщує цей код у окремому файлі або в спеціально позначеному тегу. Файли коду, як правило, мають імена типу «MyPage.aspx.cs» або «MyPage.aspx.vb», а файл сторінки — MyPage.aspx (таке ж ім'я, як і у файла сторінки (ASPX), але з розширенням, що визначає сторінку мови). Ця практика використовується у Visual Studio та інших IDE. Також, у форматі веб-додатків, pagename.aspx.cs є частковим класом, який пов'язаний з файлом pagename.designer.cs. Файл дизайнера — це файл, який автоматично створюється з ASPX-сторінки, і дозволяє розробнику посилатись на компоненти сторінки ASPX зі сторінки CS без необхідності їх оголошувати вручну, як це було в попередніх версіях ASP.NET. Використовуючи цей стиль програмування, розробник пише код, що відповідає на різні події, такі як завантаження сторінки або натискання елемента керування, а не лише на процедурний перегляд документа.

Code-behind model ASP.NET відрізняється від класичного ASP, оскільки він заохочує розробників створювати додатки, відокремлюючи презентацію та зміст. Теоретично, це дозволить веб-дизайнерам, наприклад, зосередити більше уваги на розмітці дизайну, звертаючи менше уваги на порушення програмного коду, який його запускає. Це схоже на відокремлення контролера від представлення в рамках model–view–controller (MVC).

.NET використовує технологію рендерингу «відвіданих композитів» («visited composited»). Під час компіляції файл шаблону складається в код ініціалізації, який створює дерево керування (композит), що представляє вихідний шаблон. Літерний текст переходить в екземпляри класу Literal control, а елементи керування сервером в екземпляри специфічного класу керування. Код ініціалізації поєднується з кодом, написаним користувачем (зазвичай шляхом збірки декількох часткових класів), і переходить до специфічного класу сторінки.

Фактичні запити на сторінку обробляються у кілька кроків. По-перше, під час ініціалізації створюється екземпляр класу сторінки та виконується код ініціалізації. Це створює початкове дерево керування, яке маніпулюється методами сторінки в наступних кроках. Оскільки кожен вузол у дереві являє собою елемент керування, представлений як екземпляр класу, код може змінювати структуру дерева, а також маніпулювати властивостями / методами окремих вузлів. Нарешті, під час етапу візуалізації користувач (відвідувач) має відвідати кожен вузол у дереві, вимагаючи, щоб кожен вузол сам використовував методи відвідувача. Вихідний HTML надсилається клієнту.

Після того, як запит було оброблено, екземпляр класу сторінки відкидається, а разом з ним і все дерево керування. Це викликає труднощі у програмістів-початківців ASP.NET, які звертаються до членів екземпляру класу, які відкидаються з кожним циклом запиту / відповіді сторінки.

Хоча ASP.NET бере свою назву від старої технології Microsoft ASP, вона значно від неї відрізняється. Microsoft повністю перебудувала ASP.NET, ґрунтуючись на Common Language Runtime (CLR), який є основою всіх застосунків Microsoft .NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у пакет .NET Framework (C#, Visual Basic.NET, і JScript.NET). ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію, і так далі.

2.2 ASP.NET Web API

Засіб Web API засноване на додаванні в додаток ASP.NET MVC Framework контролера спеціального виду. Цей різновид контролерів, яка називається контролером API, володіє двома характеристиками:

1. Методи дій повертають об'єкти моделей, а не об'єкти типу ActionResult.
2. Методи дій вибираються на основі HTTP-методу, використовуваного в запиті.

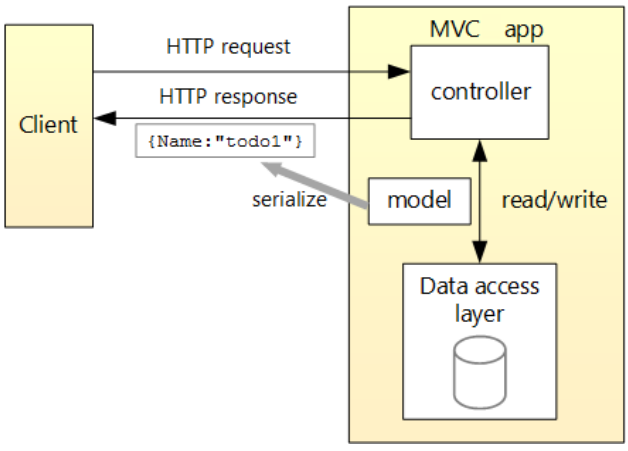

Рисунок 2.1 – Структура додатку, написаного за допомогою технології ASP.NET Web API

Об'єкти моделей, які повертаються методом дії контролера API, кодуються в форматі JSON і відправляються клієнту. Контролери API призначені для доставки веб-служб даних, тому вони не підтримують уявлення, компонування або будь-які інші засоби, які застосовувалися для генерації HTML-розмітки в прикладі створеного додатки.

Відсутність можливості у контролера API генерувати HTML-розмітку з уявлень є причиною, по якій в односторінкових додатках комбінуються стандартні прийоми ASP.NET MVC Framework з Web API. Інфраструктура ASP.NET MVC Framework виконує кроки, необхідні для доставки HTML-вмісту

користувачеві (включаючи аутентифікацію, авторизацію, вибір і візуалізацію уявлення). Після того, як HTML-вміст доставлено в браузер, запити Ajax, що генеруються містяться всередині кодом JavaScript, будуть оброблятися контролером Web API.

В звичайних контролерах можна створювати методи дій, які повертають дані JSON для підтримки Ajax, але контролер API пропонує альтернативний підхід. Цей підхід передбачає відокремлення дій, що відносяться до даних, від дій, пов'язаних з поданням, і робить створення універсальних програм Web API швидким і простим.

2.3ASP.NET MVC

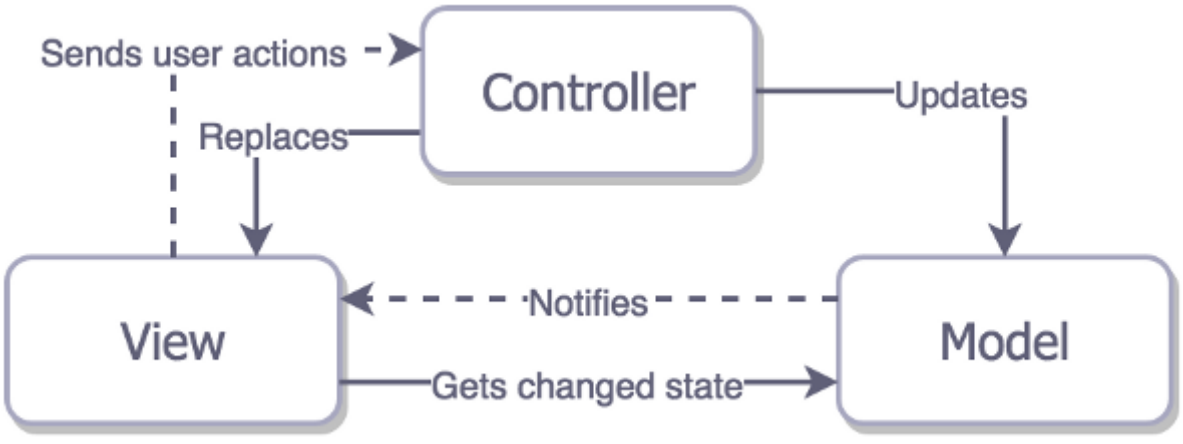

Рисунок 2.2 – Принцип роботи технології ASP.NET MVC

ASP.NET MVC - це структура веб-додатків, розроблена корпорацією Майкрософт, яка реалізує модель моделі-перегляду-контролера (MVC). Це програмне забезпечення з відкритим кодом, крім компонента ASP.NET Web Forms, який є власністю.

У більш пізніх версіях ASP.NET, ASP.NET MVC, ASP.NET Web API і ASP.NET Web Pages (платформа, що використовує тільки сторінки Razor) об'єднаються в уніфікований MVC 6.

На основі ASP.NET, ASP.NET MVC дозволяє розробникам програмного забезпечення створювати веб-додатки як композицію з трьох ролей: Model, View і Controller. Модель MVC визначає веб-додатки з 3 логічними шарами:

1. Модель (бізнес-рівень)
2. Вид (відображувальний шар)
3. Контролер (контроль входу)

Модель представляє стан конкретного аспекту програми. Контролер обробляє взаємодію та оновлює модель, щоб відобразити зміну стану програми, а потім передає інформацію до перегляду. Вид приймає необхідну інформацію від контролера і надає інтерфейс користувача для відображення цієї інформації.

У квітні 2009 року вихідний код ASP.NET MVC був випущений під ліцензією Microsoft Public License.

Каркас ASP.NET MVC - це легка, високотомізована структура презентацій, яка інтегрована з існуючими функціями ASP.NET. Деякі з цих інтегрованих функцій - це сторінки-шаблони та аутентифікація на основі членства. зборів. "

Каркас ASP.NET MVC об'єднує моделі, перегляди та контролери, використовуючи контракти на основі інтерфейсу, тим самим дозволяючи кожному компоненту тестуватися незалежно.

Двигуни перегляду, що використовуються в рамках ASP.NET MVC 3 і MVC 4, є Razor і Web Forms. Обидва двигуни перегляду є частиною системи MVC 3. За замовчуванням, механізм перегляду в рамках MVC використовує сторінки Razors .cshtml і .vbhtml або Web Forms .aspx для розробки макетів сторінок інтерфейсу користувача, на яких складаються дані. Однак можуть використовуватися різні двигуни перегляду. Крім того, замість типової моделі зворотного зв'язку ASP.NET Web Forms, будь-які взаємодії направляються до контролерів за допомогою механізму маршрутизації ASP.NET. Перегляди можуть відображатися на різних URL-адресах.

2.4 ASP.NET Core., переваги відносно попередніх версій технології ASP.NET

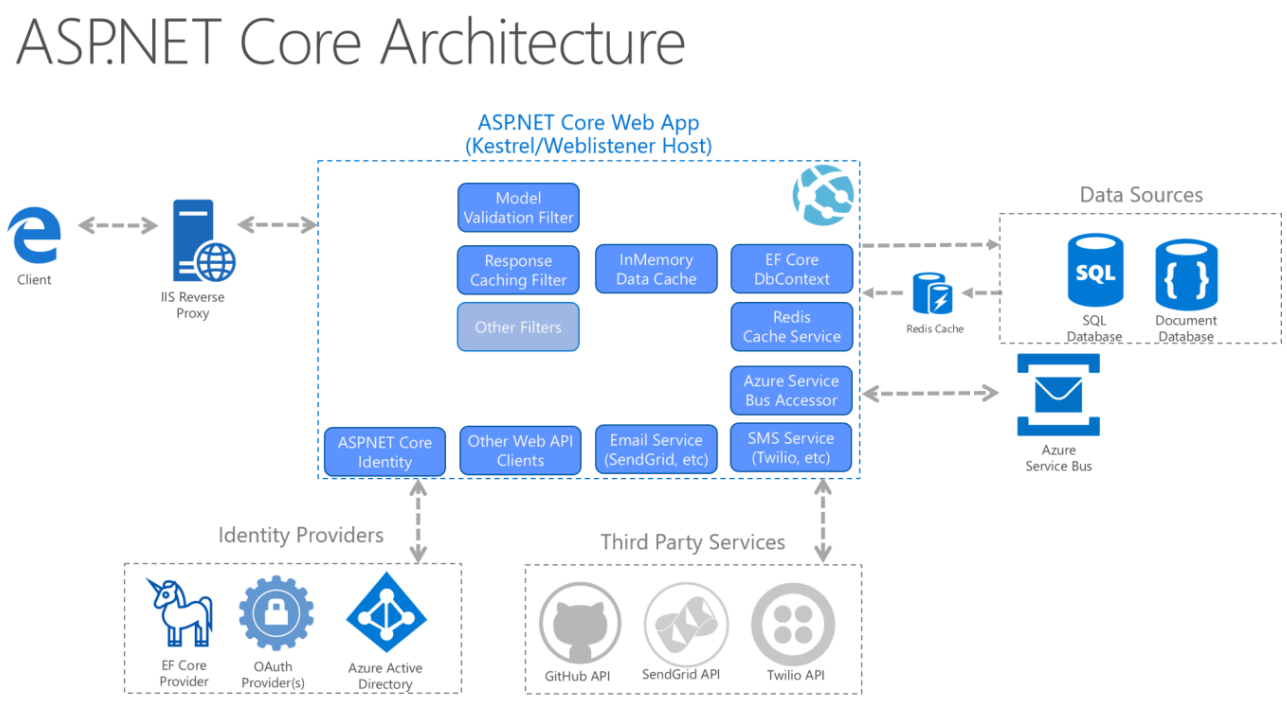

Рисунок 2.3 – Структура додатків, написаних за допомогою технології ASP.NET Core

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в травні 2018 року побачила версія ASP.NET Core 2.1, яка власне і охоплена в поточному керівництві.

ASP.NET Core тепер повністю є opensource-фреймворком. Всі вихідні файли фреймворку доступні на GitHub.

ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS X, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, але тепер вже ми не обмежені тільки цією операційною системою. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Хоча ASP.NET Core переважно націлене на використання .NET Core, але фреймворк також може працювати і з повною версією фреймворка .NET.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле.

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій.

Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом C#.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати

вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки ми можемо використовувати останні випуски Visual Studio, починаючи з версії Visual Studio 2015. Крім того, ми можемо створювати додатки в середовищі Visual Studio Code, яка є крос-платформної і може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

2.5 Методи написання веб-застосунків на ASP.NET Core

Кожен великий проект використовує паттерни для кращої структуризації коду та його підтримки в майбутньому. Паттерн – це “рекомендація” розробникам для написання якісного коду, тобто це не панацея і будь-який розробник в праві відмовитися від використання паттернів.

Розглянемо основні із них, що використовуються при написанні програм з використанням технології ASP.NET Core:

1. Розділення структури додатку на шари, використання так званої багатошарової архітектури (n-tier architecture).

2. Використання паттерна MVC(модель, представлення, контроллер), даний паттерн використовується ще й у веб-додатках для розшарування логіки, графічного інтерфейсу та доступу до даних;
3. Використання паттерна DI (вставка залежностей), класи в програмі не повинні залежати від конкретної реалізації, а від інтерфейсів, а самі залежності повинен вставляти контейнер;
4. Репозиторій(repository) – даний патерн використовується як метод доступу до об‘єктів бази даних.
5. Використання принципів SOLID.

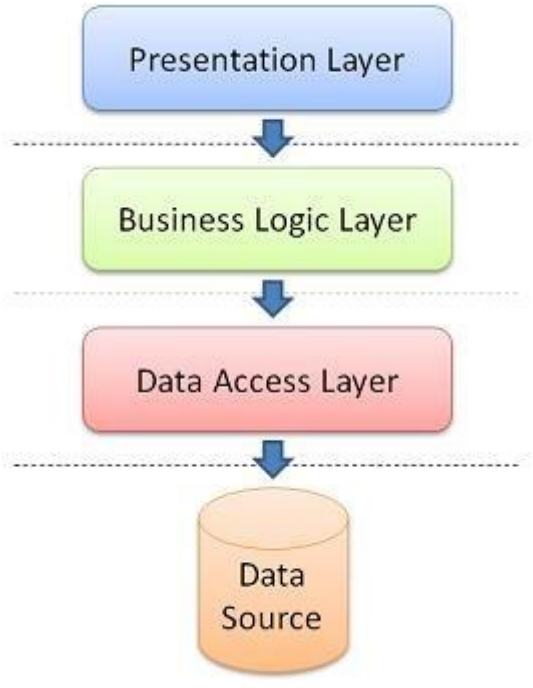

Рисунок 2.4 – Приклад багаторівневої архітектури

Багаторівнева архітектура[20] – клієнт-серверна архітектура, в якій розділяються функції представлення, обробки і зберігання даних. Найбільш поширеною різновидом багаторівневої архітектури є трирівнева архітектура.

N-рівнева архітектура додатка надає модель, по якій розробники можуть створювати гнучкі і повторно-використовуваних програм. Поділяючи додаток на рівні абстракції, розробники набувають можливість внесення змін в якийсь

певний шар, замість того, щоб переробляти все додаток цілком. Трирівнева архітектура зазвичай складається з рівня уявлення, рівня бізнес логіки і рівня зберігання даних.

Хоча поняття шару і рівня найчастіше використовуються як взаємозамінні, багато сходяться на думці, що між ними все-таки є відмінність. Різниця полягає в тому, що шар - це механізм логічного структурування компонентів, з яких складається програмне рішення, в той час як рівень - це механізм фізичного структурування інфраструктури системи. Трьохрівневе рішення легко може бути розгорнуто на єдиному рівні, такому як персональна робоча станція.

Цей паттерн передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

Інжекція залежностей - це метод, за допомогою якого один об'єкт (або статичний метод) постачає залежності іншого об'єкта. Залежність - це об'єкт, який можна використовувати (служба). Ін'єкція - це передача залежності на залежний об'єкт (клієнт), який буде використовувати його. Послуга входить до складу держави клієнта. Передача послуги клієнту, а не дозвіл клієнту побудувати або знайти послугу, є основною вимогою шаблону.

Метою ін'єкції залежностей є досягнення розділення проблем будівництва та використання об'єктів.

Ін'єкція залежності є однією з форм більш широкої техніки інверсії контролю. Як і в інших формах інверсії контролю, інжекція залежностей підтримує принцип інверсії залежностей. Клієнт делегує відповідальність за надання своїх залежностей зовнішньому коду (інжектору). Клієнту не дозволяється викликати інжекторний код, це ін'єкційний код, який конструює служби і викликає клієнта, щоб ввести їх. Це означає, що клієнтський код не повинен знати про код ін'єкції, про те, як побудувати послуги або навіть які фактичні послуги він використовує; клієнт повинен знати лише про внутрішні інтерфейси послуг, оскільки вони визначають, як клієнт може користуватися послугами. Це розділяє обов'язки використання та будівництва.

Існує три загальні засоби для клієнта для прийняття ін'єкції залежностей: інжекція засновника, інтерфейсу та конструктора. Інжектор сетера і конструктора відрізняється головним чином, коли вони можуть бути використані. Інжекція інтерфейсу відрізняється тим, що дається можливість контролювати власну ін'єкцію. Кожен з них вимагає, щоб окремий код будівництва (інжектор) брав на себе відповідальність за введення клієнта та його залежності один від одного.

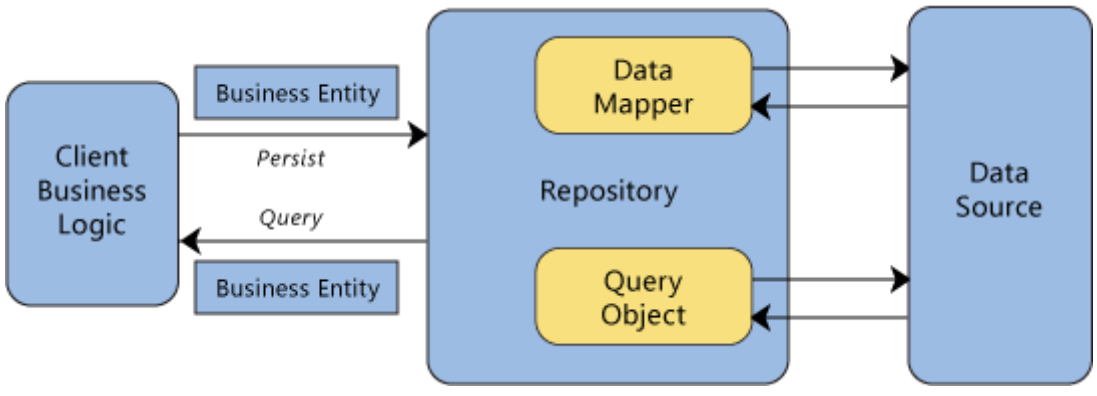

Рисунок 2.5 – Призначення паттерна Repository

Посередником між рівнями області визначення і розподілу даних (domain and data mapping layers), використовуючи інтерфейс, схожий з колекціями для доступу до об'єктів області визначення.

Система зі складною моделлю області визначення може бути спрощена за допомогою додаткового рівня, наприклад Data Mapper, який би ізолював об'єкти від коду доступу до БД. В таких системах може бути корисним додавання ще одного шару абстракції поверх шару розподілу даних (Data Mapper), в якому б був зібраний код створення запитів. Це стає ще більш важливим, коли в області визначення безліч класів або при складних, важких запитах. У таких випадках додавання цього рівня особливо допомагає скоротити дублювання коду запитів.

Патерн Repository є посередником між шаром області визначення і шаром розподілу даних, працюючи, як звичайна колекція об'єктів області визначення. Об'єкти-клієнти створюють опис запиту декларативно і направляють їх до об'єкта-сховища (Repository) для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, прихований в об'єкті Repository, подбає про відповідних операціях в непомітно для розробника. У двох словах, патерн Repository инкапсулирует об'єкти, представлениие в сховище даних і операції, вироблені над ними, надаючи більш об'єктно-орієнтоване уявлення реальних даних. Repository також має на меті досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних.

SOLID – це абревіатура складена з перших літер п’яти базових принципів об’єктно-орієнтованого програмування дизайну, запропонована Робертом Мартіном.

Принципи SOLID використовуються для дизайну та розробки таких програмних систем, які, з великою ймовірністю, зможуть тривалий час розвиватися, розширятися і підтримуватися.

Принцип єдиного обов'язка (Single Responsibility Principle) можна сформулювати так: у класу повинна бути тільки одна причина для зміни. Під обов'язком тут розуміється набір функцій, які виконують єдине завдання. Суть цього принципу полягає в тому, що клас повинен виконувати одну єдину задачу. Весь функціонал класу повинен бути цілісним, володіти високою зв'язністю (high

cohesion). Конкретне застосування принципу залежить від контексту. В даному випадку важливо розуміти, як змінюється клас. Якщо клас виконує кілька різних функцій, і вони змінюються окремо, то це якраз той випадок, коли можна застосувати принцип єдиної обов'язки. Тобто іншими словами, у класу кілька причин для зміни.

Принцип відкритості/закритості (Open/Closed Principle) можна сформулювати так: об’єкти програми повинні бути відкриті для розширення, але закриті для зміни. Суть цього принципу полягає в тому, що система повинна бути побудована таким чином, що всі її подальші зміни повинні бути реалізовані за допомогою додавання нового коду, а не зміни вже існуючого.

Принцип підстановки Лісков (Liskov Substitution Principle) являє собою деякий керівництво по створенню ієрархій успадкування. Повинна бути можливість замість базового типу підставити будь-який його підтип. Фактично принцип підстановки Лісков допомагає чіткіше сформулювати ієрархію класів, визначити функціонал для базових і похідних класів і уникнути можливих проблем при застосуванні поліморфізму.

Принцип поділу інтерфейсів (Interface Segregation Principle) відноситься до тих випадків, коли класи мають "жирний інтерфейс", тобто занадто роздутий інтерфейс, не всі методи і властивості якого використовуються і можуть бути затребувані. Таким чином, інтерфейс вийдуть занадто надмірний або "жирним". Принцип поділу інтерфейсів можна сформулювати так: клієнти не повинні вимушено залежати від методів, якими не користуються. При порушенні цього принципу клієнт, який використовує певний інтерфейс з усіма його методами, залежить від методів, якими не користується, і тому виявляється сприйнятливий до змін в цих методах. У підсумку ми приходимо до жорсткої залежності між різними частинами інтерфейсу, які можуть бути не пов'язані при його реалізації. В цьому випадку інтерфейс класу поділяється на окремі частини, які становлять окремі інтерфейси. Потім ці інтерфейси незалежно один від одного можуть застосовуватися і змінюватися. В результаті застосування принципу поділу

інтерфейсів робить систему слабосвязанной, і тим самим її легше модифікувати і оновлювати.

Принцип інверсії залежностей (Dependency Inversion Principle) служить для створення слабосвязанних сутностей, які легко тестувати, модифікувати і оновлювати. Цей принцип можна сформулювати наступним чином: модулі верхнього рівня не повинні залежати від модулів нижнього рівня. І ті й інші повинні залежати від абстракцій. Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

2.6 Фреймворк для роботи з базою даних Entity Framework Core

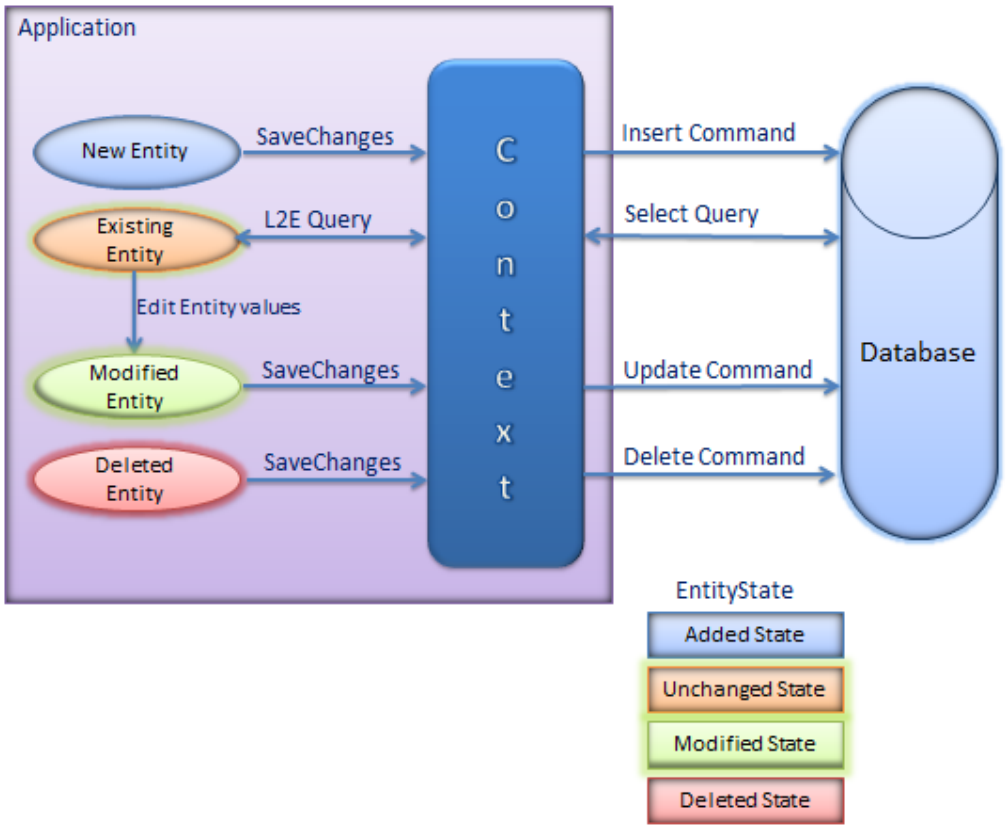

Рисунок 2.6 – Структура роботи технології EF Core

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і расширяемую технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, для SQLite, для PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, ми вирішимо змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і настройки підключення до відповідних провайдерам. А код, який безпосередньо працює з даними, отримує дані, додає їх в БД і т.д., залишиться колишнім.

Entity Framework Core багато успадкував від своїх попередників, зокрема, Entity Framework 6. У той же час треба розуміти, що EF Core - це не нова версія по відношенню до EF 6, а зовсім інша технологія, хоча в цілому принципи роботи у них будуть збігатися. Тому в рамках EF Core використовується своя система версій. Поточна версія - 2.0 була випущена в серпні 2017 року. І технологія продовжує розвиватися. Що вже є в EF Core, а що тільки планується додати, можна подивитися в роудмапе на гітхабе.

Як технологія доступу до даних Entity Framework Core може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні додатки, WPF, ASP.NET 4.6 / 4.7. Це і нові технології як UWP і ASP.NET Core. При цьому кроссплатформенная природа EF Core дозволяє задіяти її не тільки на ОС Windows, але і на Linux і Mac OS X.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людини, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік. Властивості необов'язково представляють прості дані типу int або string, але можуть також представляти і більш комплексні типи даних. І у кожної сутності може бути одна або кілька

властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому суті можуть бути пов'язані асоціативної зв'язком один-ко-многим, один-ко-одному і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core, як технології ORM, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибірку об'єктів, в тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконання запиту транслює вираження LINQ в вирази, зрозумілі для конкретної СУБД (як правило, в вирази SQL).

2.7 Технологія для роботи з базою даних ADO.NET

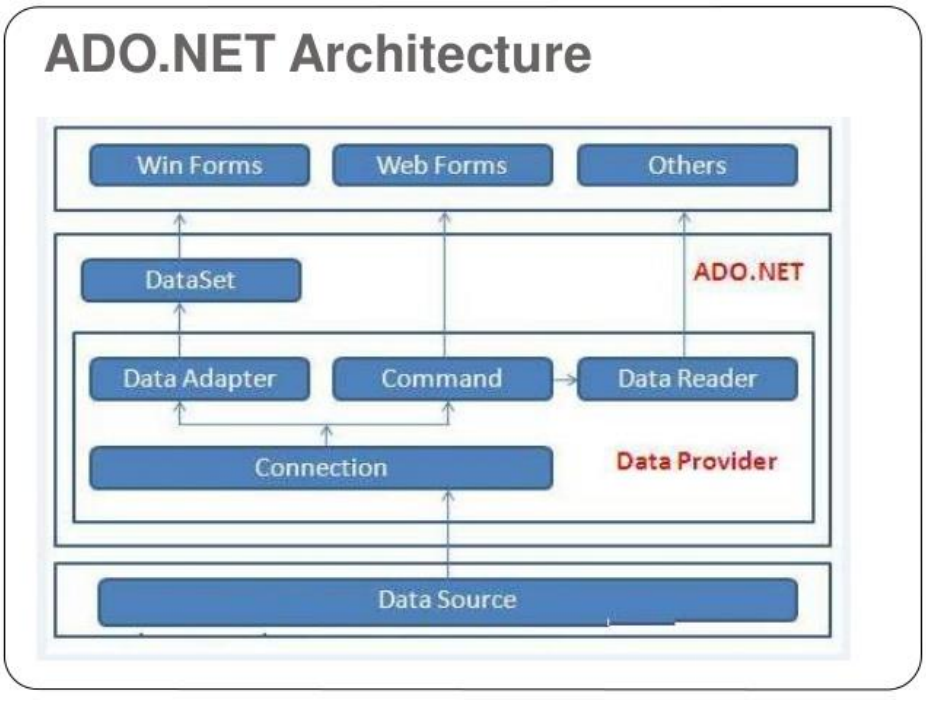

Рисунок 2.7 – Структура технології ADO.NET

Сьогодні велике значення має робота з даними. Для зберігання даних використовуються різні системи управління базами даних: MS SQL Server, Oracle, MySQL і так далі. І більшість великих додатків так чи інакше використовують для зберігання даних ці системи управління базами даних. Однак щоб здійснювати зв'язок між базою даних і додатком на C # необхідний посередник. І саме таким посередником є технологія ADO.NET.

ADO.NET надає собою технологію роботи з даними, яка заснована на платформі .NET Framework. Ця технологія являє нам набір класів, через які ми можемо відправляти запити до баз даних, встановлювати підключення, отримувати відповідь від бази даних і виробляти ряд інших операцій.

Причому важливо зазначити, що систем управління баз даних може бути безліч. У своїй сутності вони можуть відрізнятися. MS SQL Server, наприклад, для створення запитів використовує мову T-SQL, а MySQL і Oracle застосовують мову PL-SQL. Різні системи баз даних можуть мати різні типи даних. Також можуть

відрізнятися якісь інші моменти. Однак функціонал ADO.NET побудований таким чином, щоб надати розробникам уніфікований інтерфейс для роботи з самими різними СУБД.

Основу інтерфейсу взаємодії з базами даних в ADO.NET представляє обмежене коло об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. За допомогою об'єкта Connection відбувається установка підключення до джерела даних. Об'єкт Command дозволяє виконувати операції з даними з БД. Об'єкт DataReader зчитує отримані в результаті запиту дані. Об'єкт DataSet призначений для зберігання даних з БД і дозволяє працювати з ними незалежно від БД. І об'єкт DataAdapter є посередником між DataSet і джерелом даних. Головним чином, через ці об'єкти і буде йти робота з базою даних.

Однак щоб використовувати один і той же набір об'єктів для різних джерел даних, необхідний відповідний провайдер даних. Власне через провайдер даних в

ADO.NET і здійснюється взаємодія з базою даних. Причому для кожного джерела даних в ADO.NET може бути свій провайдер, який власне і визначає конкретну реалізацію вищевказаних класів.

За замовчуванням в ADO.NET є наступні вбудовані провайдери:

1. Провайдер для MS SQL Server
2. Провайдер для OLE DB (Надає доступ до деяких старих версій MS SQL Server, а також до БД Access, DB2, MySQL і Oracle)
3. Провайдер для ODBC (провайдер для тих джерел даних, для яких немає своїх провайдерів)
4. Провайдер для Oracle
5. Провайдер EntityClient. Провайдер даних для технології ORM Entity Framework
6. Провайдер для сервера SQL Server Compact 4.0

Крім цих провайдерів, які є вбудованими, існує також безліч інших, призначених для різних баз даних, наприклад, для MySQL.

Функціонально класи ADO.NET можна розбити на два рівня: підключений і відключений. Кожен провайдер даних .NET реалізує свої версії об'єктів Connection, Command, DataReader, DataAdapter і ряду інших, який складають підключений рівень. Тобто за допомогою них встановлюється підключення до БД і виконується з нею взаємодію. Як правило, реалізації цих об'єктів для кожного конкретного провайдера в своїй назві мають префікс, який вказує на провайдер.

Інші класи, такі як DataSet, DataTable, DataRow, DataColumn і ряд інших складають відключений рівень, так як після отримання даних в DataSet ми можемо працювати з цими даними незалежно від того, чи встановлено підключення чи ні. Тобто після отримання даних з БД додаток може бути відключено від джерела даних.

2.8 Висновок до розділу 2

В даному розділі було розглянуто основні можливості ASP.NET для побудови додатків а також основні фреймворки для роботи з базами даних Entity Framework Core та ADO.NET. Також було розглянуто версії ASP.NET, такі як ASP.NET Web API, ASP.NET MVC та ASP.NET Core. Окрім можливостей вищезазначених фреймворків, було розглянуто історії створення даних технологій, їх розвиток та призначення.

3. РОЗРОБКА ТА АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

3.1 Проектування інформаційної системи

Основні можливості програми показано на діаграмі прецедентів (рис. 3.1):

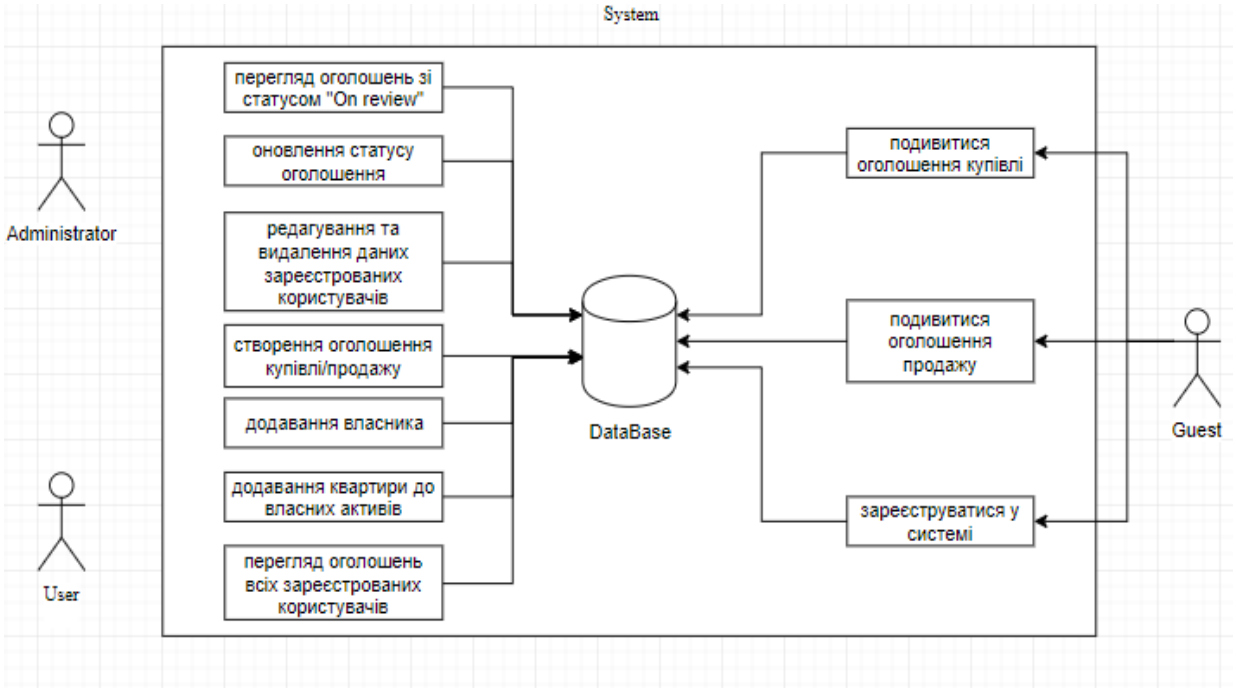

Рисунок 3.1 – Діаграма прецедентів

Для розроблення вищезазначеної системи було вирішено використати об’єктно-орієнтовану мову програмування C# та використовувати наступні технології як: .NET Framework, ASP.NET Core, EF, SSMS, Swagger.

.NET вибрано через те що він надає надзвичайно великий функціонал і є кросплатформенним. Дивлячись на те що тип програми це WebApi використання цього коду можливе усюди. Бо найголовніше в таких проектах саме логіка, яку в подальшому можна портувати будь-куди: веб-сайт, смартфон, персональний комп’ютер. Також великою перевагою цієї мови є те, що реалізований так званий «збірник сміття», а саме Garbage collector[19]. Основна його робота полягає в звільненні задіяної, проте вже не потрібної пам’яті додатку. Кожного разу при створенні нового об’єкту загальномовне середовище виділяє пам’ять для об’єкта

з керованої кучі. Проте пам’ять не безмежна. В решті решт збірник сміття повинен виконати збір для того щоб звільнити частину задіяної пам’яті.

Як раніше згадувалося, в даній роботі було використано наступні технології: .NET Framework, ASP.NET Core, EF, SSMS, Swagger. Під час аналізу отриманої задачі постало питання розгортки серверу. Для цього передбачено IIS Express. Оскільки основним завданням було розроблення саме Api, в проекті для показу функціоналу у браузері використовується бібліотека Swagger.

Для побудови запитів до БД обрано мову SQL. На сьогодні це найбільш загальновживане рішення. Оскільки вона перевірена часом. Обрано СУБД SSMS для доступу та управління базою даних.

Розглянемо діаграму уявлення (рис. 3.4, 3.5). На рисунку зображено шість контроллерів (класи верхнього рівня): AccountController, ApartmentOwnersController, ApartmentsController, RentAnnouncementsController, SaleAnnouncementsController, AnnouncementsController.

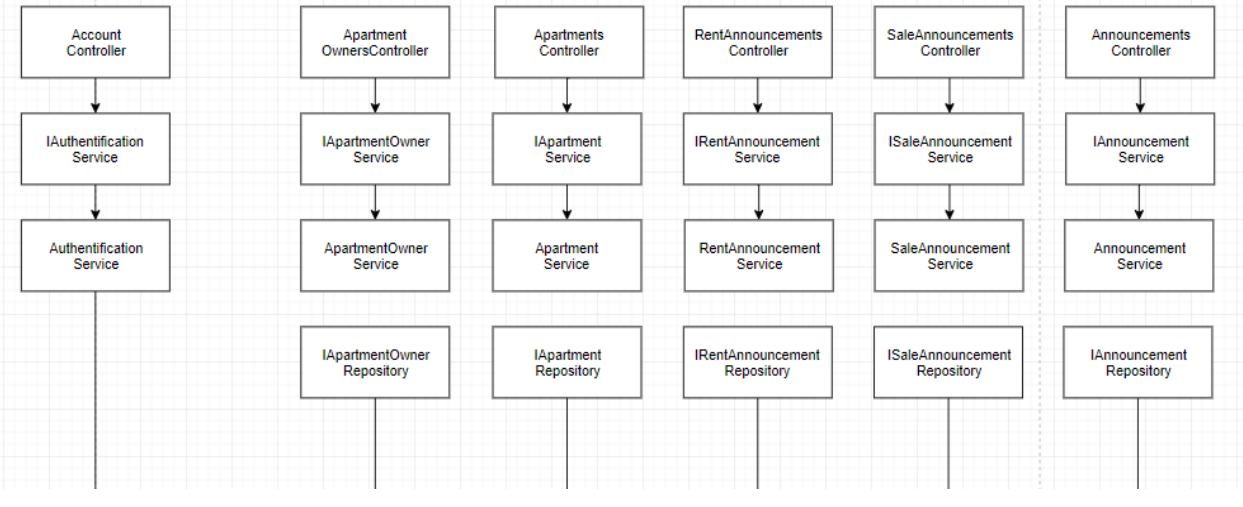

Рисунок 3.2 – Діаграма уявлення

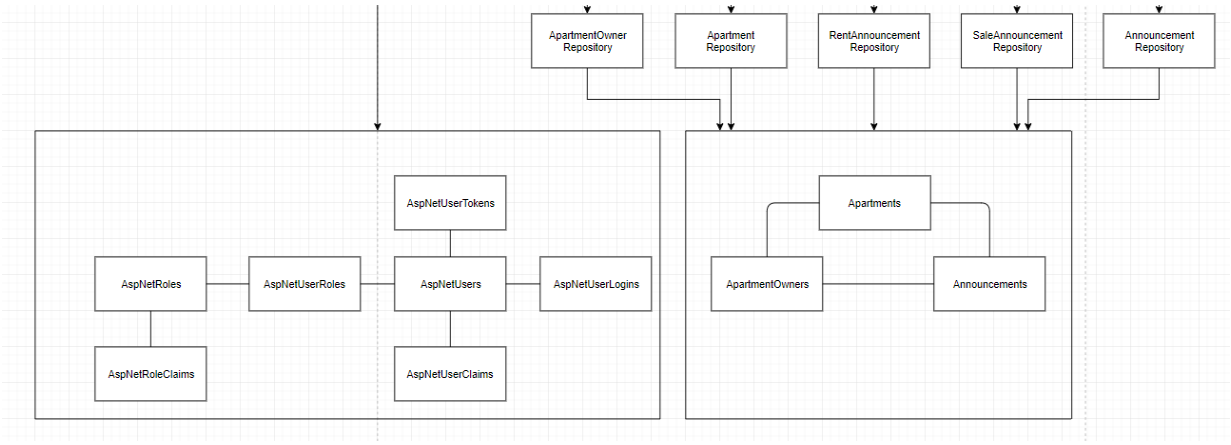

Рисунок 3.3 – Діаграма уявлення

Ці класи відповідають на запити користувачів. Всі вони знаходяться в презентаційному рівні. Для подальшої обробки запиту вони йдуть до рінвя бізнес-логіки, де звертаються до інтерфейсів своїх сервісів, які в свою чергу звертаються до самих сервісів. Далі йде посилання на конкретний інтерфейс репозиторію, який посилається на конкретний репозиторій.

Потім всі репозиторії насилають запити до БД. Така структура реалізована для того щоб забезпечити більшу гнучкість і модульність.

3.2 Створення структури бази даних

Для проектування бази даних спочатку потрібно проаналізувати завдання. В нас дуже багато інформації, яку потрібно структурувати і логічно розподілити.

Перше на що треба звернути увагу, це те, що в нас буде дві БД. Оскільки одна нам потрібна для роботи системи, а інша для автентифікації та авторизації.

Для проектування бази даних спочатку маємо проаналізувати предметну область задачі. У базі даних потрібно зберігати багато інформації про структуру системи. У рішенні проекту для бази даних є окремий проект під назвою EstateEgency.DAL. Так як однією з основних вимог була розробка логічного і правильного найменування namespace та папок, було розбито проект на папки для зручного пошуку необхідних класів.

Загалом маємо в першу чергу було створено абстрактний клас, який потім наслідують усі сутності. Це клас Entity, який знаходиться у папці Base, яка в свою чергу лежить у Entities. Це є базовий клас, який описує унікальний ідентифікатор для кожної сутності. Зробивши його абстрактним, ми, по-перше, реалізовуємо принципи SOLID, по-друге, прибираємо дублювання коду.

Далі в тій самій папці Entities створюємо логічні сутності для роботи.

Клас Announcement – це клас оголошення. Він зберігає у собі посилання на апартамент, посилання на унікальний ідентифікатор апартаменту, власника квартири, його ідентифікатор, статус огляду оголошення і дату створення. Для прикладу прикріплено фрагмент діаграми бази даних, на якому описано цей клас (рис. 3.4):

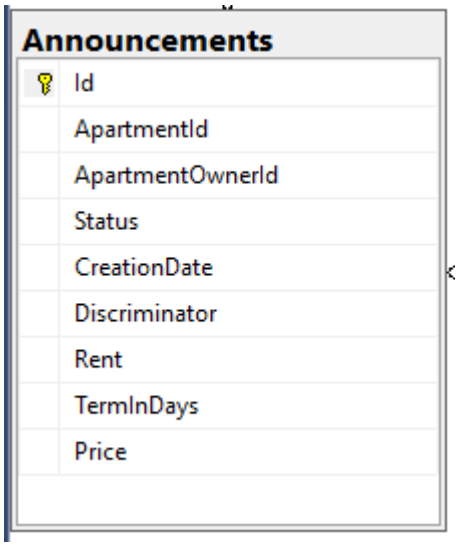

Рисунок 3.4 – клас Announcement

Клас Apartment – це клас квартири. У ньому зберігається інформація, яка може знадобитися користувачу для додаткової інформації про житло. Там є посилання на власника, його ідентифікатор, всі його оголошення, а далі опис квартири як такої. А саме – адреса, кількість кімнат, поверх, квадратура та опис її «наповнення» і інфраструктури навколо. Для прикладу прикріплено фрагмент діаграми бази даних, на якому описано цей клас (рис 3.5):

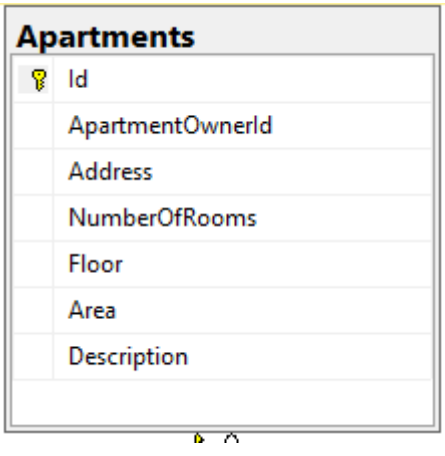

Рисунок 3.5 – клас Apartments

Останнім класом для умовного каркасу виступає ApartmentOwner. Це інформація про власника апартаменту. Вона має в собі посилання на всі оголошення цієї людини (купівлю/продаж), всі апартаменти, та особисті дані. Такі як прізвище, ім’я, телефонний номер та електронна адреса для контактування (рис 3.6):

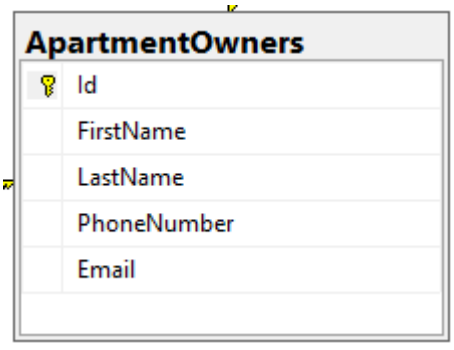

Рисунок 3.6 – клас ApartmentOwners

Далі для логічної структури системи було додано ще два класи. Перший це – RentAnnouncement. Клас для того щоб здавати в оренду. Він є нащадком класу Announcement, тобто реалізує всі його поля. В нього додано кількість днів, на які буде винаймано квартиру та вартість.

Другий клас – SaleAnnouncement. Він розроблений для випадку купівлі квартири. Також є нащадком класу Announcement, проте в ньому додано лише одне поле – ціна.

Загалом маємо таку архітектуру бази даних (див. додаток Б).

Як можна бачита на дотку Б, реалізовано зв’язки один до багатьох.

Другою базою даних виступає EstateAgencyAuthentificationDb. Вона зберігає в собі всіх користувачів, саме зареєстрованих у системі. Адміністраторів системи та звичайних користувачів.

Як бачимо вона є більш складною в проетуванні. Як відомо Entity Framework Core самостійно не реалізовує зв'язки багато до багатьох. Рішенням в такій ситуації виступає створення додаткової таблиці, яка буде пов’язувати дані.

3.3 Структура проекту

Для розробки системи було вирішено використовувати трирівневу архітектуру.

Presentation layer (рівень презентації) – це частина системи, якою керує і користується споживач. На цьому рівні є інтерфейс користувача разом з розробленою системою для отримання даних від нього. Так наприклад у технології ASP.NET MVC на цьому рівні розміщені візуальні компоненти які являють собою інтерфейс користувача (стилі, статичні сторінки html, javascript), так само і уявлення, контролери, об’єкти контексту запиту. Presentation layer реалізовано у проекті EstateEgancy.API[21].

Business layer (рівень бізнес-логіки) – тримає в собі набір компонентів, які відповідають за обробку отриманих від рівня презентації даних, реалізує всю необхідну логіку додатку, усі розрахунки, взаємодіє з базою даних та передає на рівень презентації результати обробки. Business layer реалізовано у проекті EstateEgancy.BLL[22].

Data Access layer (равень доступу до даних) – зберігає моделі, що описують сутності, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад, клас контексту даних Entity

Framework. Тут також зберігаються репозиторії, через які бізнес логіка взаємодіє з базою. Data Access layer реалізовано у проекті EstateEgancy.DAL[23].

При цьому слід зауважити, що крайні равні не можуть взаємодіяти між собою, тобто рівень презентації не може напряму звернутися до бази даних. Це можливо лише через використання бізнес логіки.

Під час написання Data Access layer було також реалізовано «сховище» та «одиницю роботи».

Сховище (repository) – посередник між доменом та рівнями відображення даних, що використовує схожий на колекцію інтерфейс для доступу до об’єктів домена. Репозиторій є посередником між доменом та рівнями відображення даних, діє як колекція об’єктів домену в пам’яті. Об’єкти клієнта створюють декларативні специфікації запитів і відправляють їх в репозиторій. Об’єкти можуть додаватися та видалятися з репозиторію так само, як зі звичайної колекції. Код що представлений репозиторієм буде робити це за кулісами. Концептуально репозиторій інкапсулює набір об’єктів, що зберігаються у базі даних і операції, що над ними виконуються. Тим самим забезпечуючу більш об’єктно-орієнтоване представлення персистентності[24].

Одиниця роботи (unit of work) – підтримує список об’єктів, що зачепила бізнес-транзакція і координує виписування змін та рішення проблеми паралелізму. Коли дані з БД виймаються, дуже важливо слідкувати що саме було змінено. В протилежному випадку ці дані не будуть потім записані назад в базу. Так само повинні бути вставлені нові об’єкти, котрі були створені і видалені будь-які об’єкти, які видаляються. Можна змінювати базу даних при кожному змінення нашої об’єктної моделі, але це може призвести до дуже маленьких звернень до бази, котрі в результаті будуть дуже повільно відпрацьовувати. Окрім того потребується щоб мі мали відкриту транзакцію для всієї взаємодії, що не є логічно, якщо ми маємо бізнес-транзакцію, що охоплює декілька запитів. Ситуація може бути ще гіршою, наприклад, якщо нам потрібно слідкувати за об’єктами, котрі ми порахували, щоб не було непослідовного читання. Одиниця роботи слідкує за усім, що ви робите під час бізнес-транзакції, котра може вплинути на базу даних.

Коли ми завершимо, вона з’ясує все що потрібно зробити, щоб змінити базу в результаті нашої роботи[25].

Велику роль відіграє DTOs (Data Transfer Objects). Це об’єкти, які транспортують дані, наприклад, до різних рівнів. Можна уявити, що клієнт отримує дані, які відображаються безпосередньо з таблиці бази даних. Проте це не завжди гарна ідея. Іноді ми маємо бажання змінити форму даних, котрі відправляємо клієнту[26]. Наприклад:

– видалити циклічні посилання;

– приховати певні властивості, котрі клієнти не повинні бачити;

– викинути певні властивості, щоб зменшити розмір корисного навантаження.

Для цього можна визначити об’єкт передачі даних (DTO). DTO – це об’єкт, котрий визначає як дані будуть відправлятися у мережі.

Для прикладу наведено дані з програми (рис. 3.7, і рис. 3.8).

```
public class ApartmentDto
{
    0 references
    public int Id { get; set; }

    3 references
    public string OwnerFirstName { get; set; }
    3 references
    public string OwnerLastName { get; set; }
    3 references
    public string OwnerPhoneNumber { get; set; }
    3 references
    public string OwnerEmail { get; set; }

    0 references
    public string Address { get; set; }
    0 references
    public int NumberOfRooms { get; set; }
    0 references
    public int Floor { get; set; }
    0 references
    public double Area { get; set; }
    0 references
    public string Description { get; set; }
}
```

Рисунок 3.7 – Базова ДТО

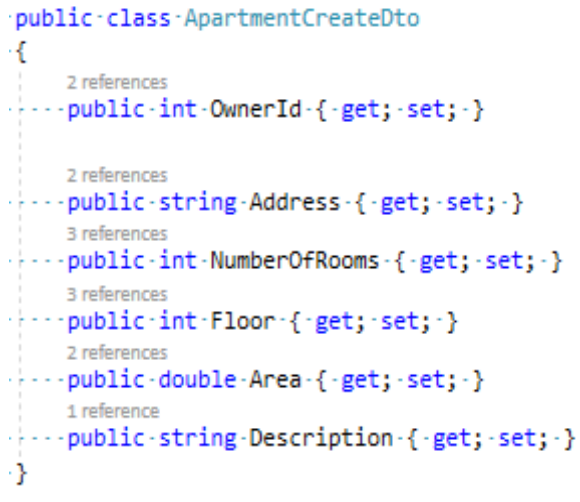

Рисунок 3.8 – ДТО для створення об’єкту

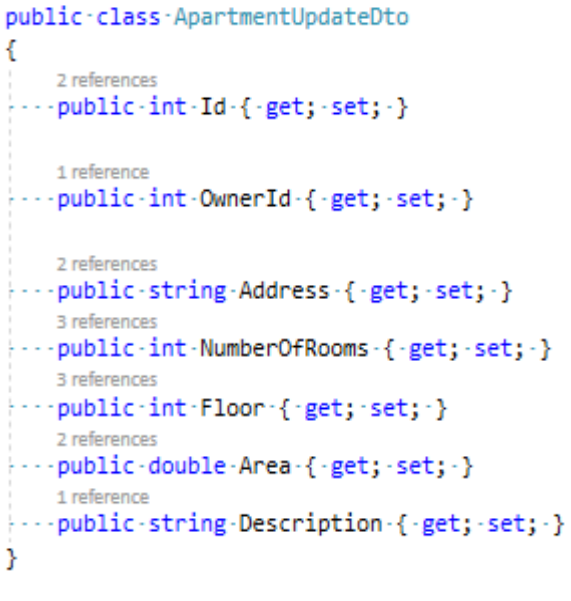

Рисунок 3.9 – ДТО для змінення об’єкту

Як бачимо, нам не потрібні усі дані моделі для конкретних методів, бо вони мають різний функціонал.

3.4 Тестовий приклад роботи програми

Перед тим як почати розділ, потрібно зазначити, що в системі є три ролі: адміністратор, користувач і гість. У всіх різні рівні доступу. В адміністратора їх найбільше. В нього немає обмежень. В той час, як в звичайного гостя їх майже

нема. Він може лише проглянути лише оголошення про продаж чи оренду. Зареєстровані користувачі можуть розміщювати свої власні об’яви, проте вони будуть недосяжні для інших до тих пір поки адміністратор не продивиться заявку і не дозволить. Змінювати і видаляти свої акаунти не можна. Це робить лише адміністратор. На рис. 3.10 представлено, в якому вигляді сайт бачить користувач:

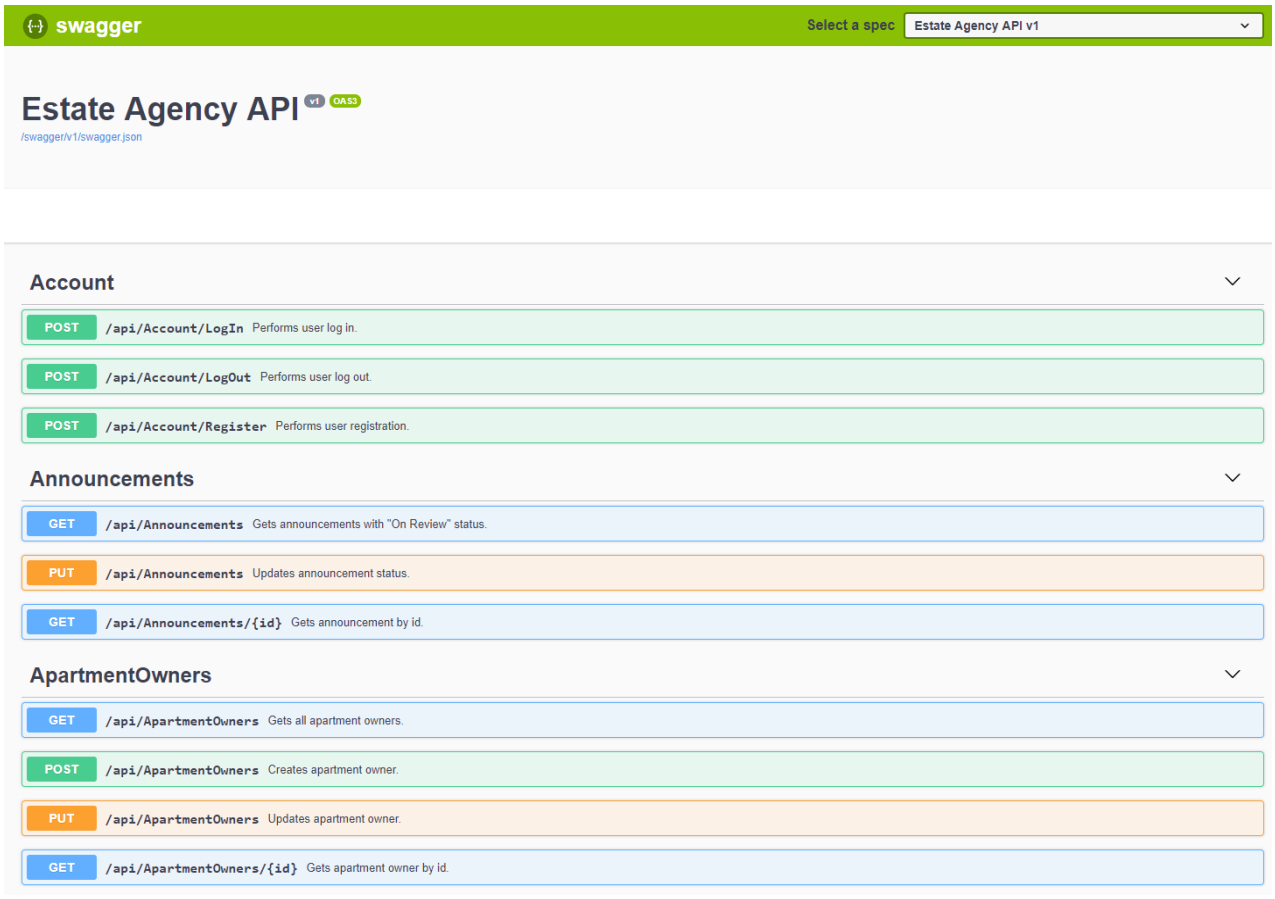

Рисунок 3.10 – Початковий вигляд сайту

У вкладці Account надається можливіть зареєструватися у системі, увійти чи вийти з неї. Всі поля з валідацією даних. Система не дозволить подальші кроки поки не будуть виконані поточні вимоги.

Для входу в акаунт необхідно ввести пошту і пароль. Ця система створена для комфортного користування. Тому дуже важливо щоб кожен користувач не боявся що його дані хтось може змінити чи зробити щось шкідливе. Якщо все введено правильно і такий користувач існує, програма видасть код 200. Проте може повернути і 400 код у випадку не коректності даних чи невалідності моделі.

Також при реєстрації передбачено контроль за вводом користувача у форму для реєстрації.

Також існує обмеження для адреси електронної пошти. Пошта повинна складатись з:

— літери англійського алфавіту великого регістру (A-Z);
— літери англійського алфавіту маленького регістру (a-z);
— цифри від 0 до 9;
— спеціальні символи такі як ~!@#$%^&*_-+=`|\(){}[]:;'<>,.?/.

Також повинна закінчуватись на, наприклад, @gmail.com[27].

Схожа ситуація і з паролем[28]. Він повинен мати:

— літери англійського алфавіту великого регістру (A-Z);
— літери англійського алфавіту маленького регістру (a-z);
— цифри від 0 до 9;
— спеціальні символи такі як ~!@#$%^&*_-+=`|\(){}[]:;'<>,.?/.

Для прикладу дивіться рис. 3.11:

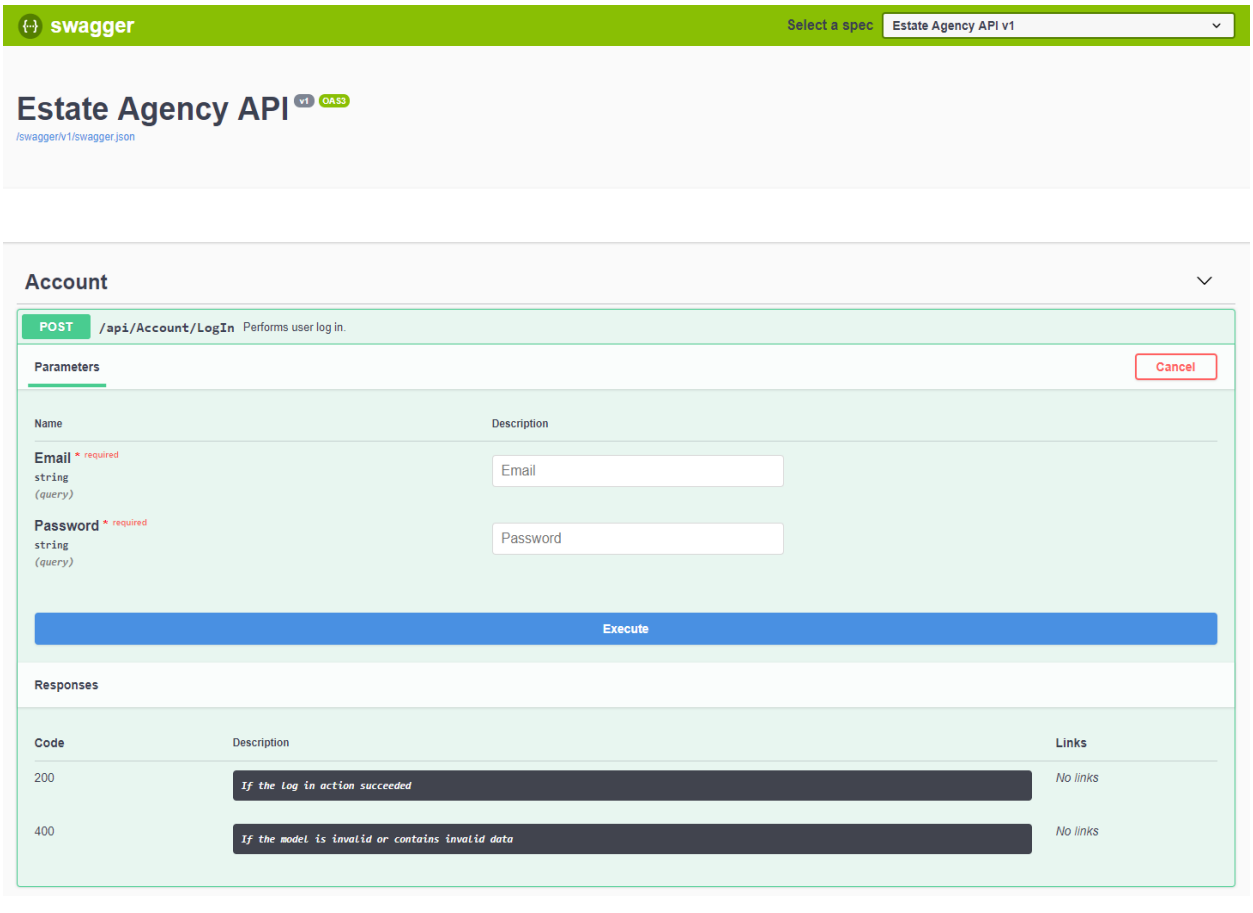

Рисунок 3.11 – LogIn

Далі наведено форму реєстрації. Для цього необхідно перейти на вкладку Account. Там буде Post метод LogIn. Натиснути кноку TryItOut. Після цього система запропонує нам ввести дані у дві існуючі форми: email та password.

У випадку якщо вхід у систему пройшов успішно, повернеться код 200.

У випадку коли модель невірна чи коли приходять невірні дані система поверне 400 код.

Також на цій вкладці також можна вийти з системи. Цю функцію забезпечує метод LogOut. Потрібно лише натиснути кнопку TryItOut. Далі слід натиснути Execute. Після завершення обробки запиту натискання на останню кнопку, користувач вийде з системи, як представлено на рис. 3.12:

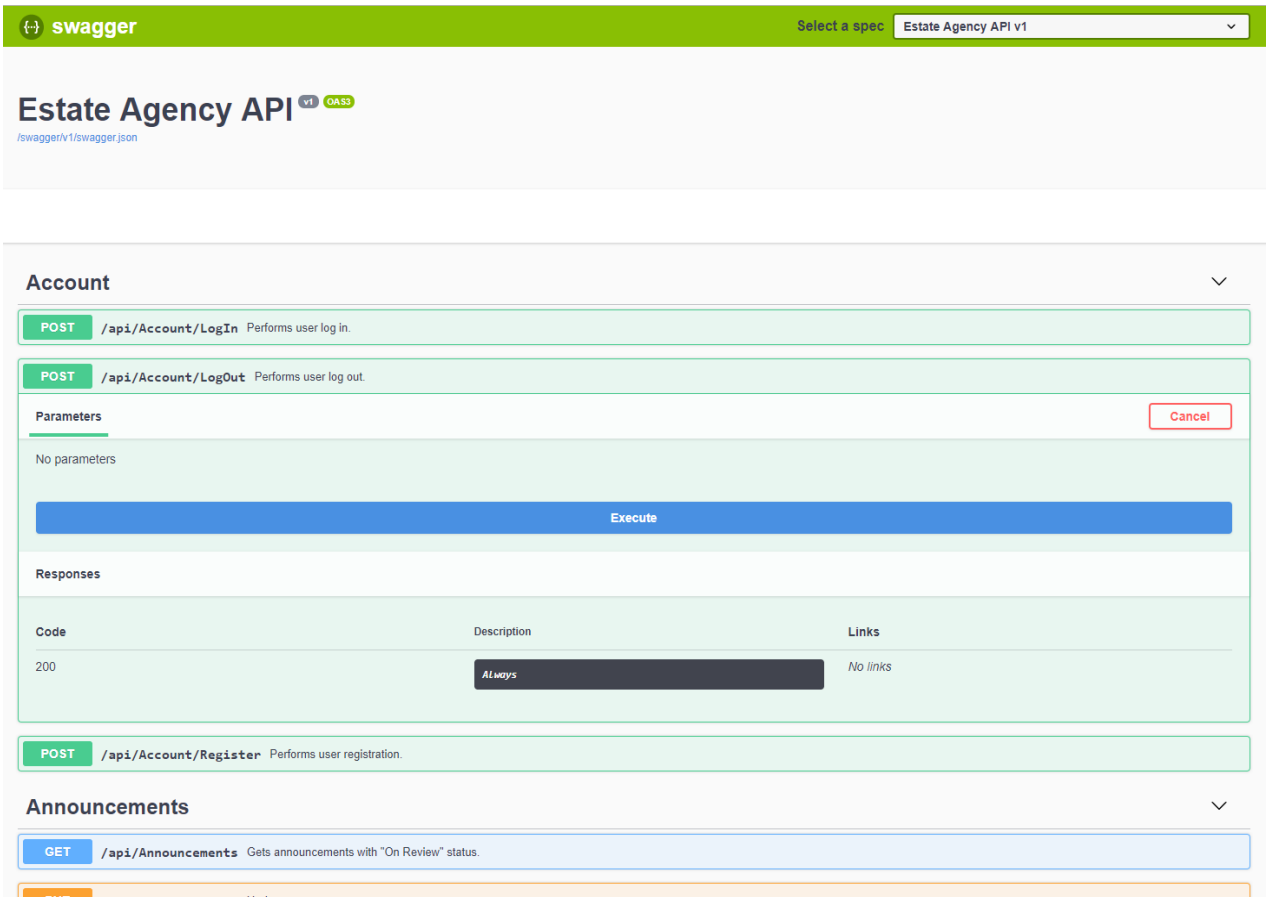

Рисунок 3.12 – LogOut

Також у системі реалізовано інструмент для реєстрації нових користувачів. Метод називається Register. Він надає нам чотири поля для заповнення інформацією, а саме:

— email – це поштова скринька, на яку в подальшому користувачі системи зможуть висилати певну інформацію. Загалом можна сказати, що це контактна інформація зареєстрованого користувача;

— password – це персональний пароль;

— passwordConfirm – поле для підтвердження паролю. Реалізовано для того щоб людина, яка реєструється в системі не забула випадково свій пароль;

— roles – поле, яке зберігає в собі значення ролі. Як відомо, у системі їх три. Проте тут ми зможемо визначити два, бо у тертьому немає потреби. Тому тут є сенс позначити “admin” та “user”.

Приклад інструменту для реєстрації нових користувачів представлено на рис. 3.13:

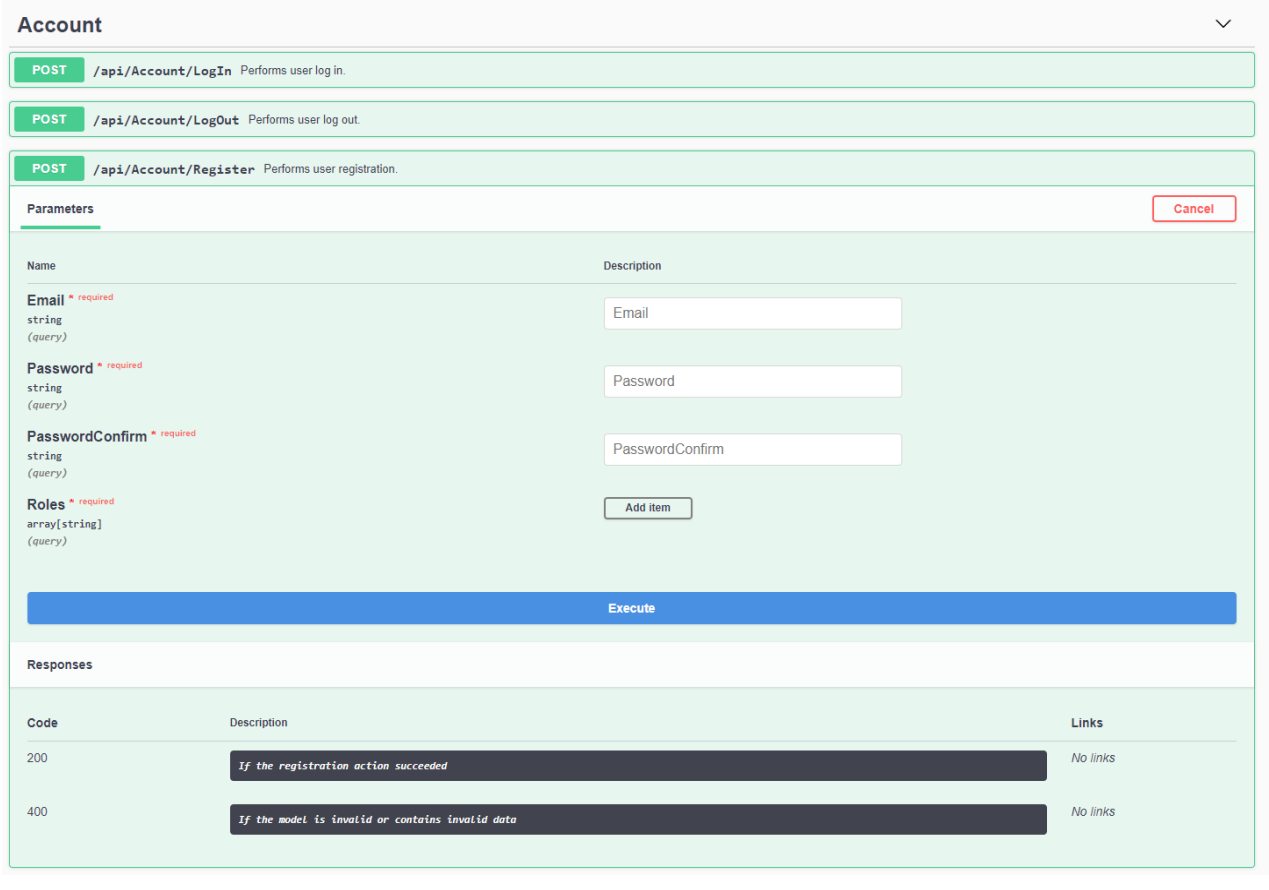

Рисунок 3.13 – Форма реєстрації

Гість може подивитися які доступні оголошення на оренду квартири. Для пошуку реалізовано декілька форм. В них є основні критерії. У випадку коли їх полишити пустими буде показано всі можливі варіанти (рис. 3.14 і рис. 3.15):

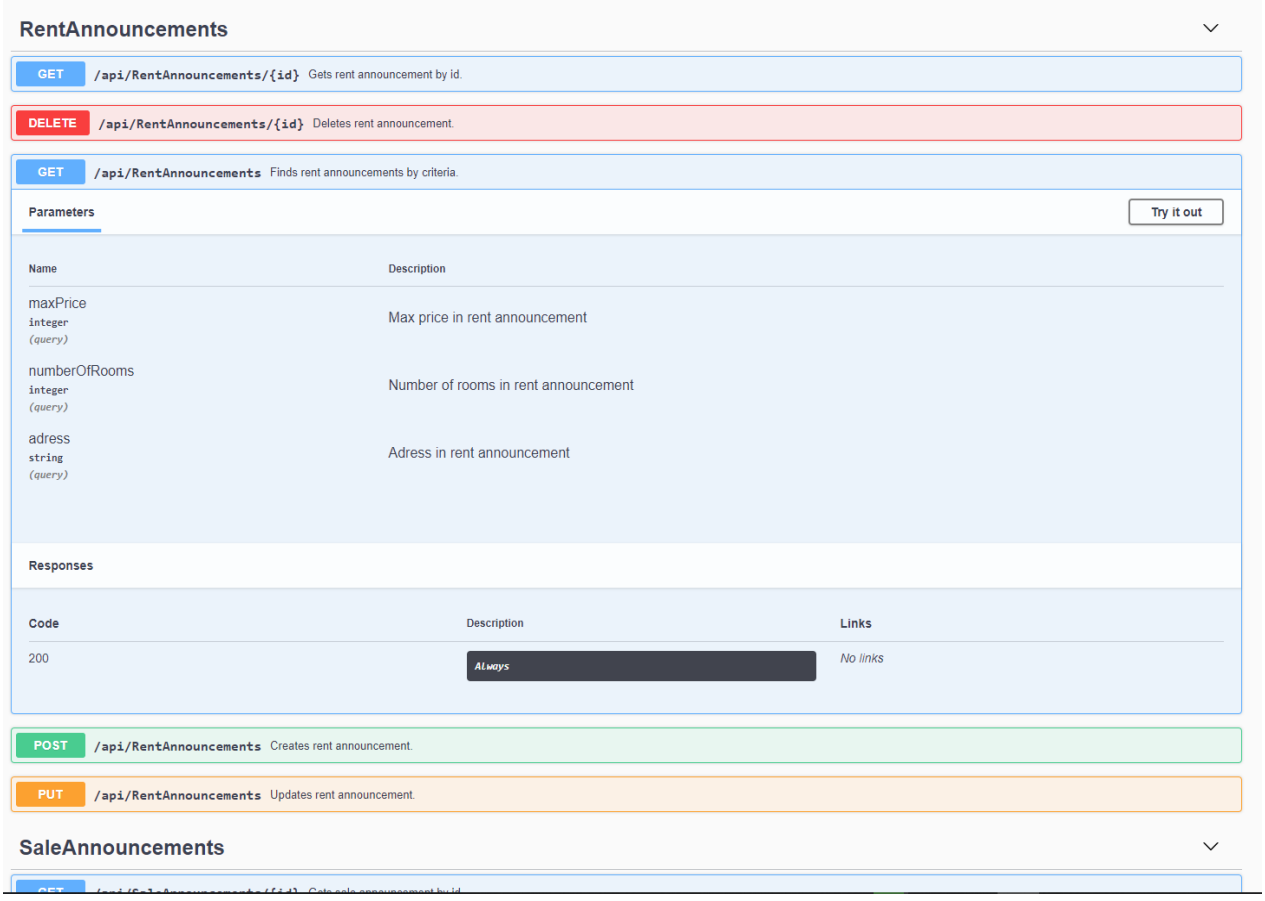

Рисунок 3.14 – Форма пошуку оренди

Загалом ми бачимо три парметри. Максимальна ціна, кількість кімнат та адресу. Тобто користувач може налаштувати ці параметри особисто під себе для найбільш точного підбору квартири за його персональними бажаннями.

Якщо лишити певні фільтри пустими, то автоматично не відпрацює, тобто видасть усі можливі варіанти з цим фільтром.

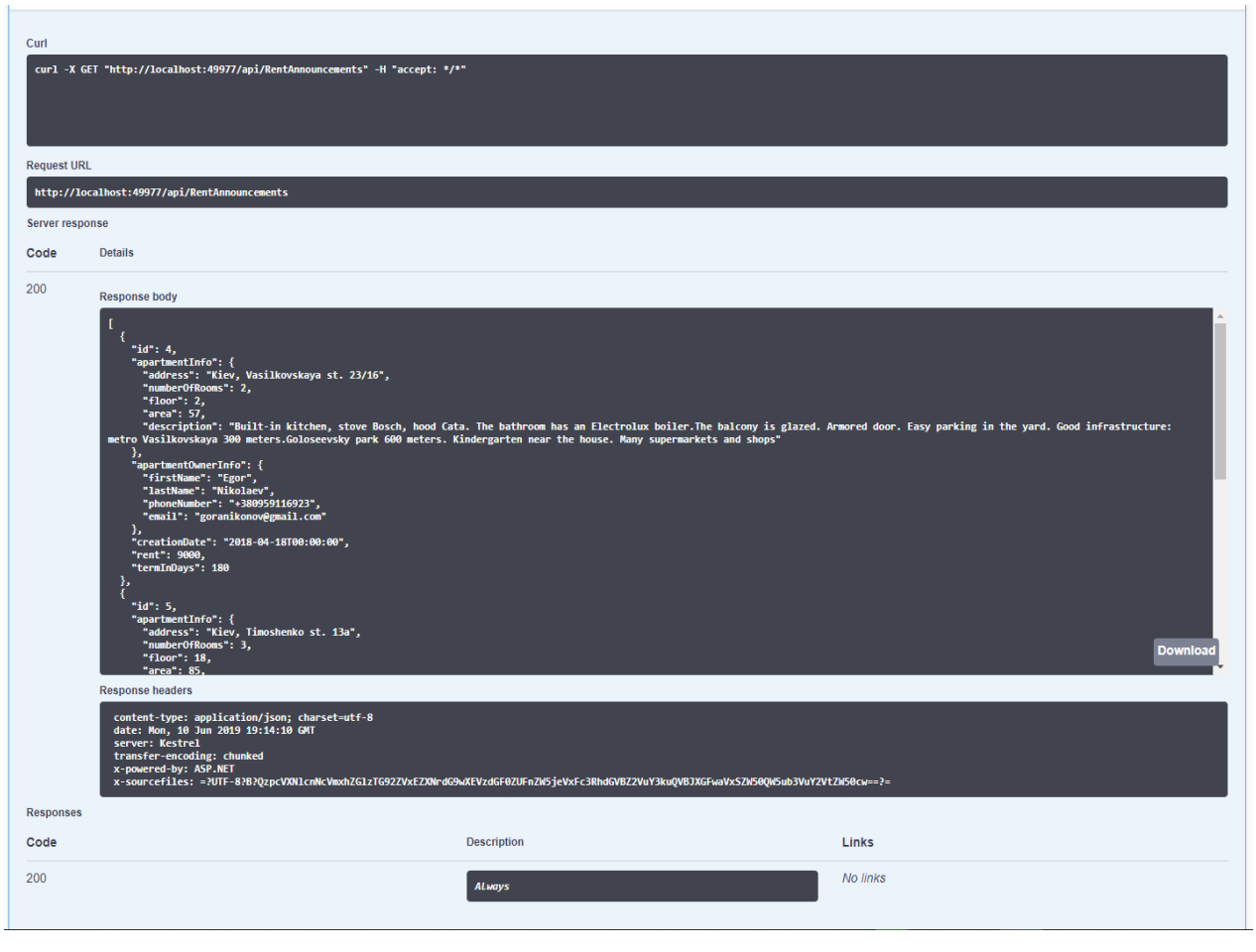

Рисунок 3.15 – Результат роботи пошуку оренди

Так само відпрацьовує пошук продажу. Там також гість може за певними критеріями відсіяти більшість варіантів, щоб знайти саме те, що йому потрібно.

Для того щоб подивитися які оголошення знаходяться у неперевіреному стані потрібно зайти під адміністратором і перейти за посиланням Announcements -> Get, як показано на рис. 3.16:

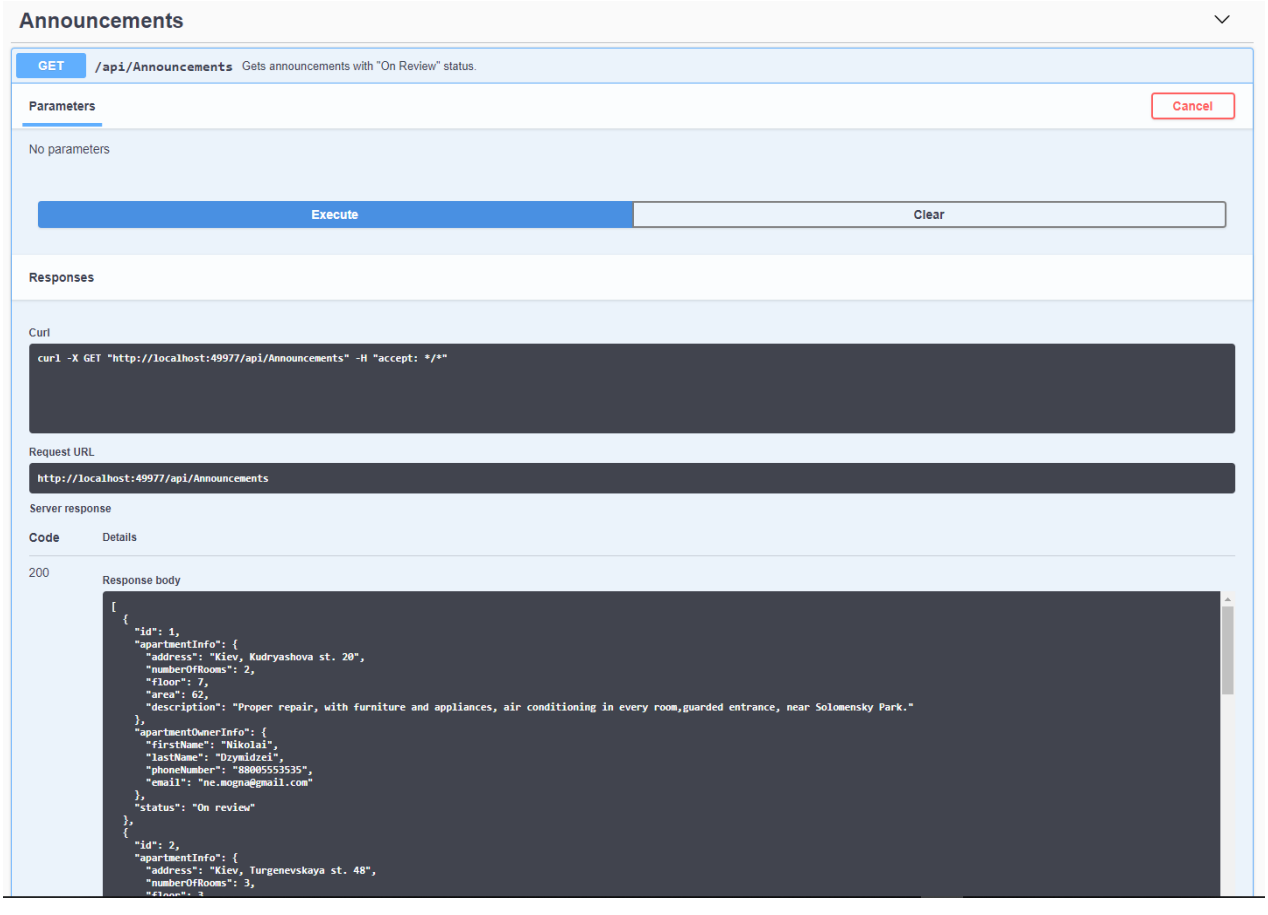

Рисунок 3.16 – Непідтверджені оголошення

Для того щоб об’ява потрапила до загального списку, який можуть переглядати потенційні покупці, необхідно щоб її прийняв адміністратор (рис. 3.15). У вкладці Announcements є метод, який повертає нам усі оголошення з позначкою “On review”. Саме це і є неперевірені об’яви.

Це свого роду контроль якості продукту. Тобто у випадку коли адміністратора все цілком влаштовує у поданому оголошенні, він змінює його статус.

Якщо адміністратора все влаштовує в поданому оголошенні і все відповідає нормам сайту, він змінює статус об'яви (рис. 3.17):

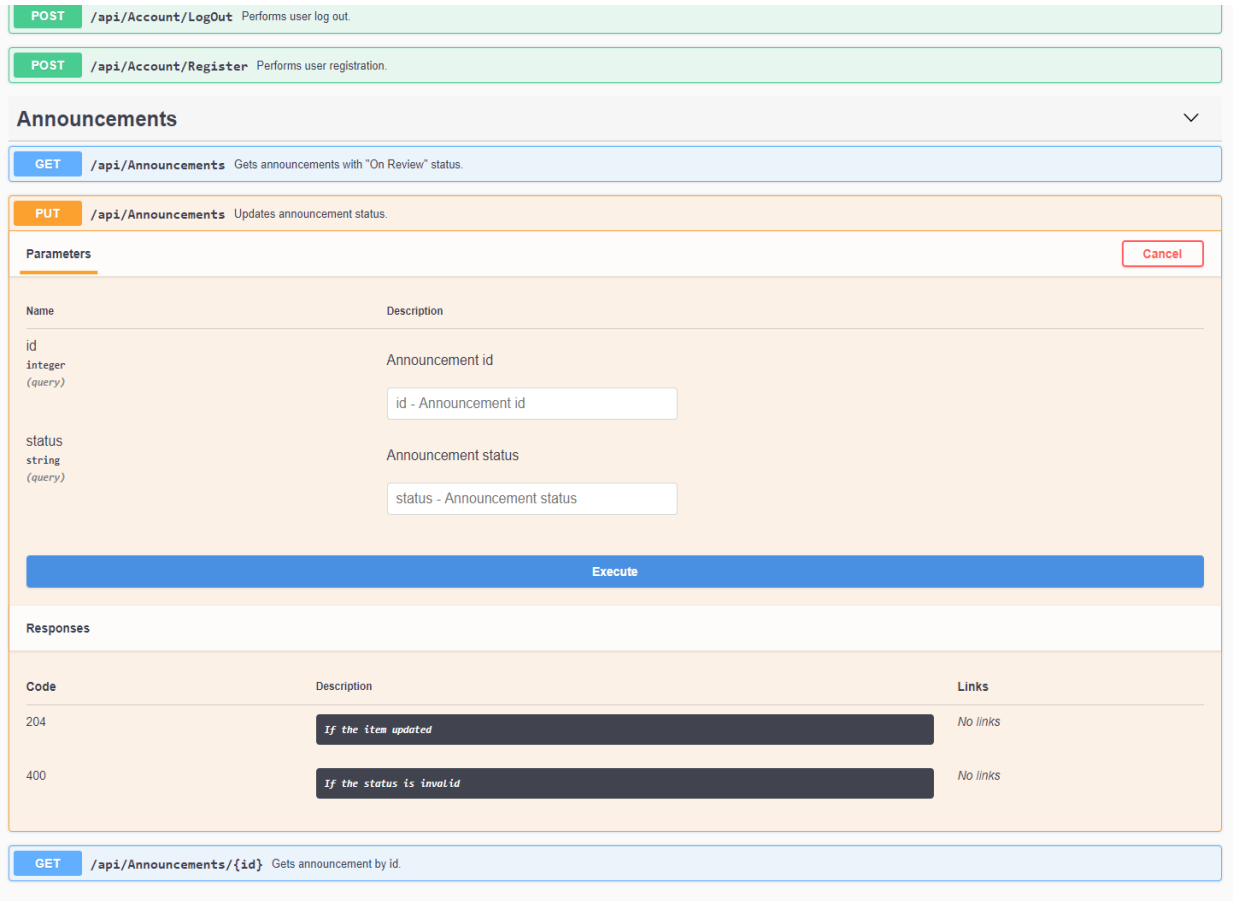

Рисунок 3.17 – Змінення статусу об'яви

Інколи бувають такі ситуації, коли необхідно подивитися не всі оголошення, статус яких “On review”, а лише одне. Для цього реалізовано окремий метод Get (рис. 3.18):

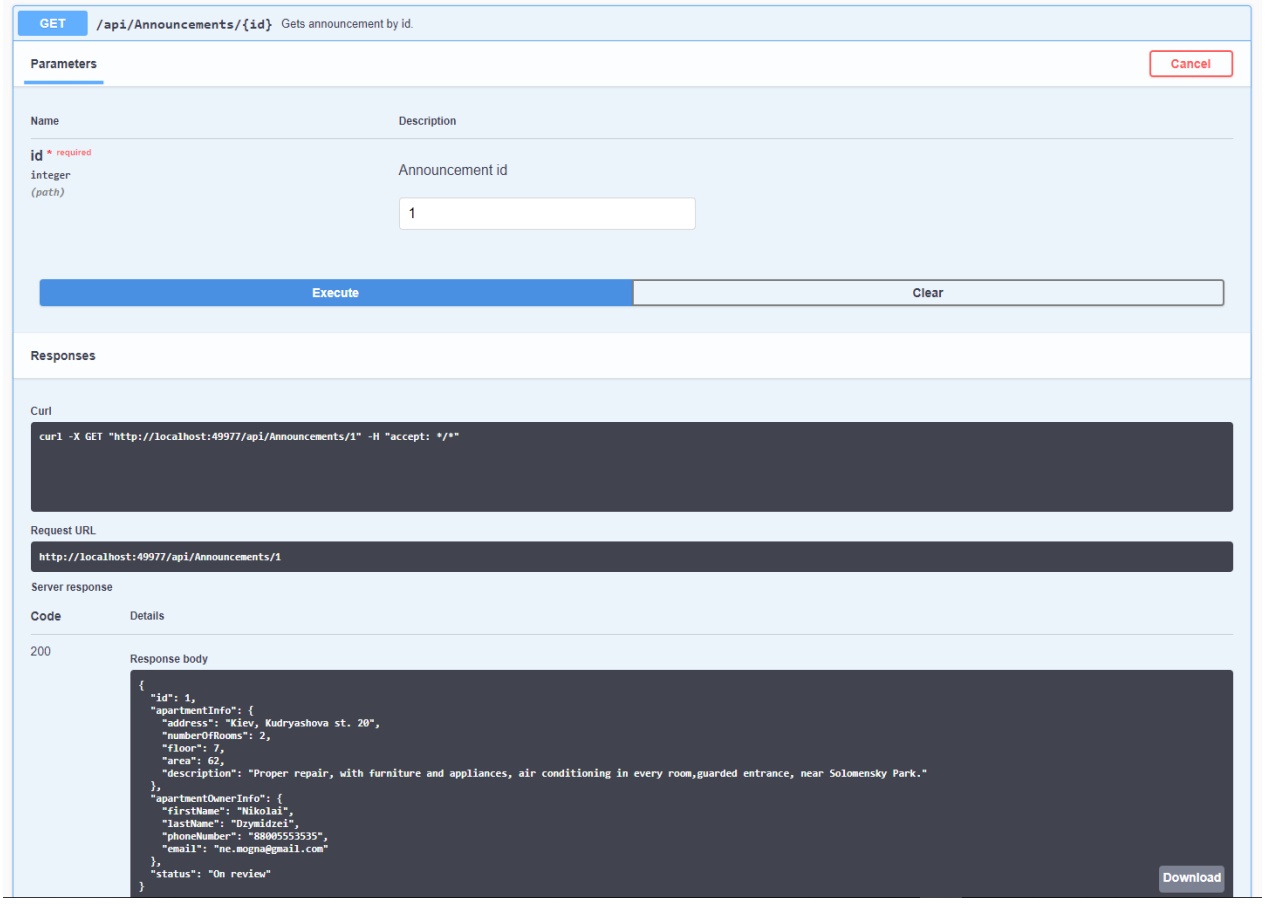

Рисунок 3.18 – Інформація по номеру оголошення

Він повертає лише одну модель, яку заповнює інформацією про сутність з вказаним унікальним ідентифікатором. Модель має у собі інформацію про апартаменти і власника. Стосовно апартаментів існують такі дані:

— address – адреса квартири;

— numberOfRooms – кількість кімнат;

— floor – поверх;

— area – площа;

— description – опис квартири як такої. Тут може бути вказане технічне оснащення квартири, переваги її розташування, наприклад, інфраструктура. Наявність поруч магазинів, аптек, шкіл, метро, дитячих садків та інше.

Також приходять такі дані про власника:

— firstName – ім’я;

— lastName – прізвище;

— phoneNumber – телефонний номер;

— email – пошта.

Пошта та адреса електронної скриньки необхідна для контактування з власником квартири та обговорення в телефонному режимі нюансів купівлі/продажу квартири.

Давайте для прикладу розглянемо власників квартир. Цей запит повертає усіх власників, які зареєстровані в системі. Серед повернутих даних є інформація щодо їх унікального ідентифікатору, прізвища, імені, номеру телефону і поштової скриньки для контактування (рис. 3.19):

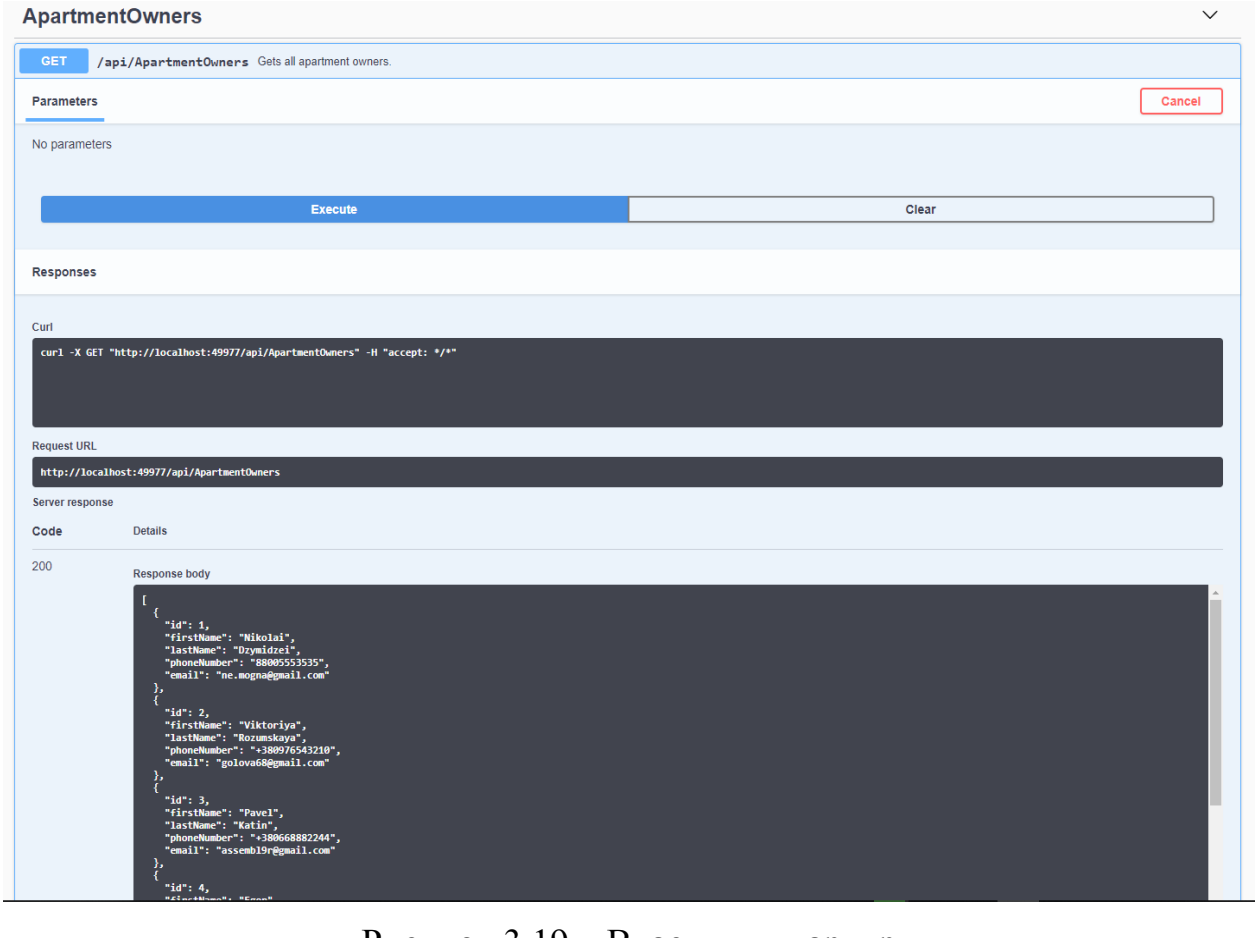

Рисунок 3.19 – Власники квартир

Пропоную також розглянути як саме відбувається механізм додавання себе як власника квартири. Для цього необхідно заповнити форму, яка приймає відповідну інформацію:

— FirstName – ім'я;

— LastName – прізвище;

— PhoneNumber;

— Email – адреса електронної пошти.

Всі ці форми з валідацією даних наведено на рис. 3.20:

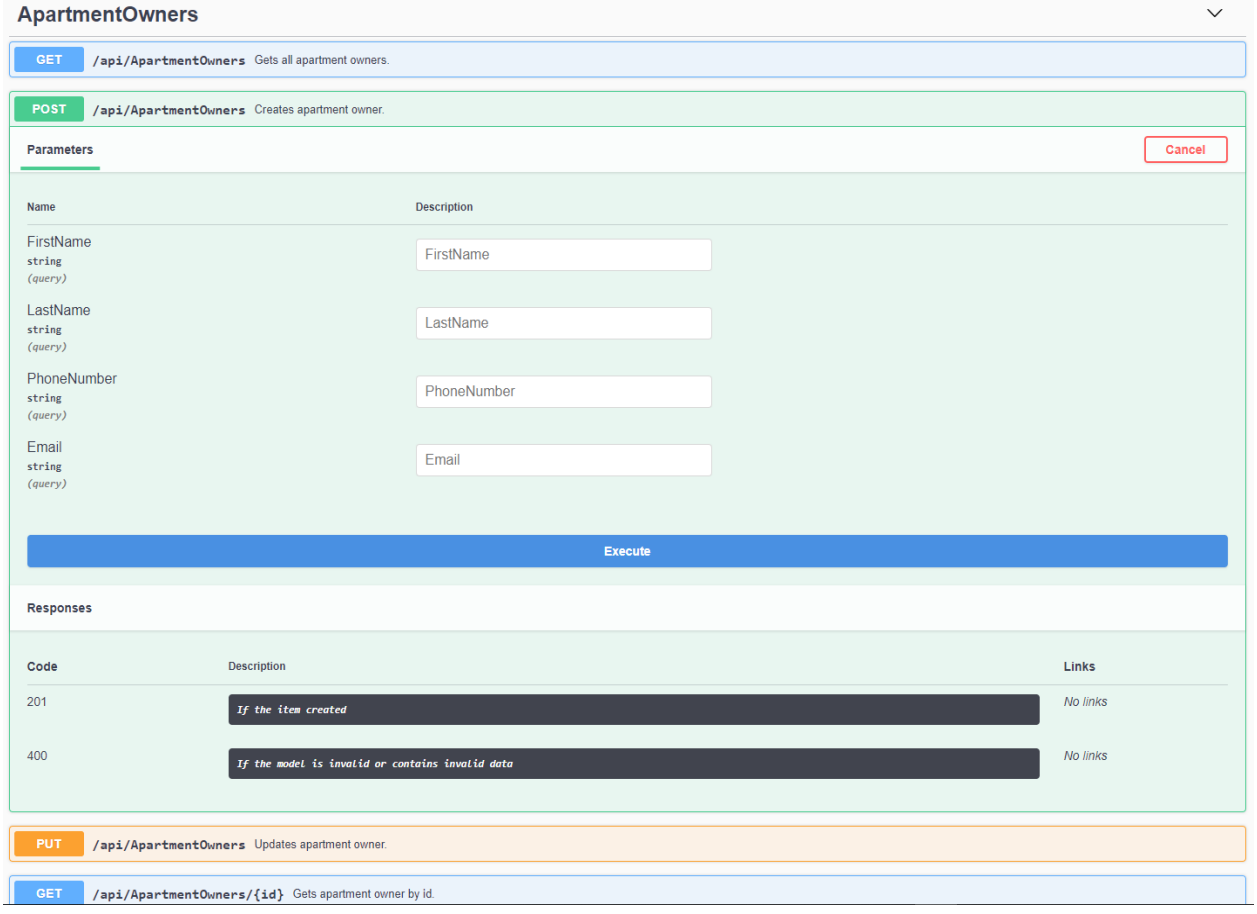

Рисунок 3.20 – Реєстрація нового власника

Після обробки запиту може бути два варіанта відповіді програми.

У випадку коли все пройшло ідеально, система поверне код 200[29].

У випадку коли було введено невірні дані, система поверне код 400. Проте там буде вказано в якому саме місці відбулася проблема. Наприклад, може бути

повідомлення невірного вводу імені, якщо лишити це поле пустим, або ж те, що телефонний номер може бути заповнений лише цифрами (0-9)[30].

Також нам може знадобитися змінення сутності власника чи його видалення. Ці функції може виконувати лише адміністратор Для видалення власника квартири необхідно вказати його унікальний ідентифікатор. Система знайде цю сутність і видалить її з бази. Приклад використання цього алгоритму можна побачити на рис. 3.21:

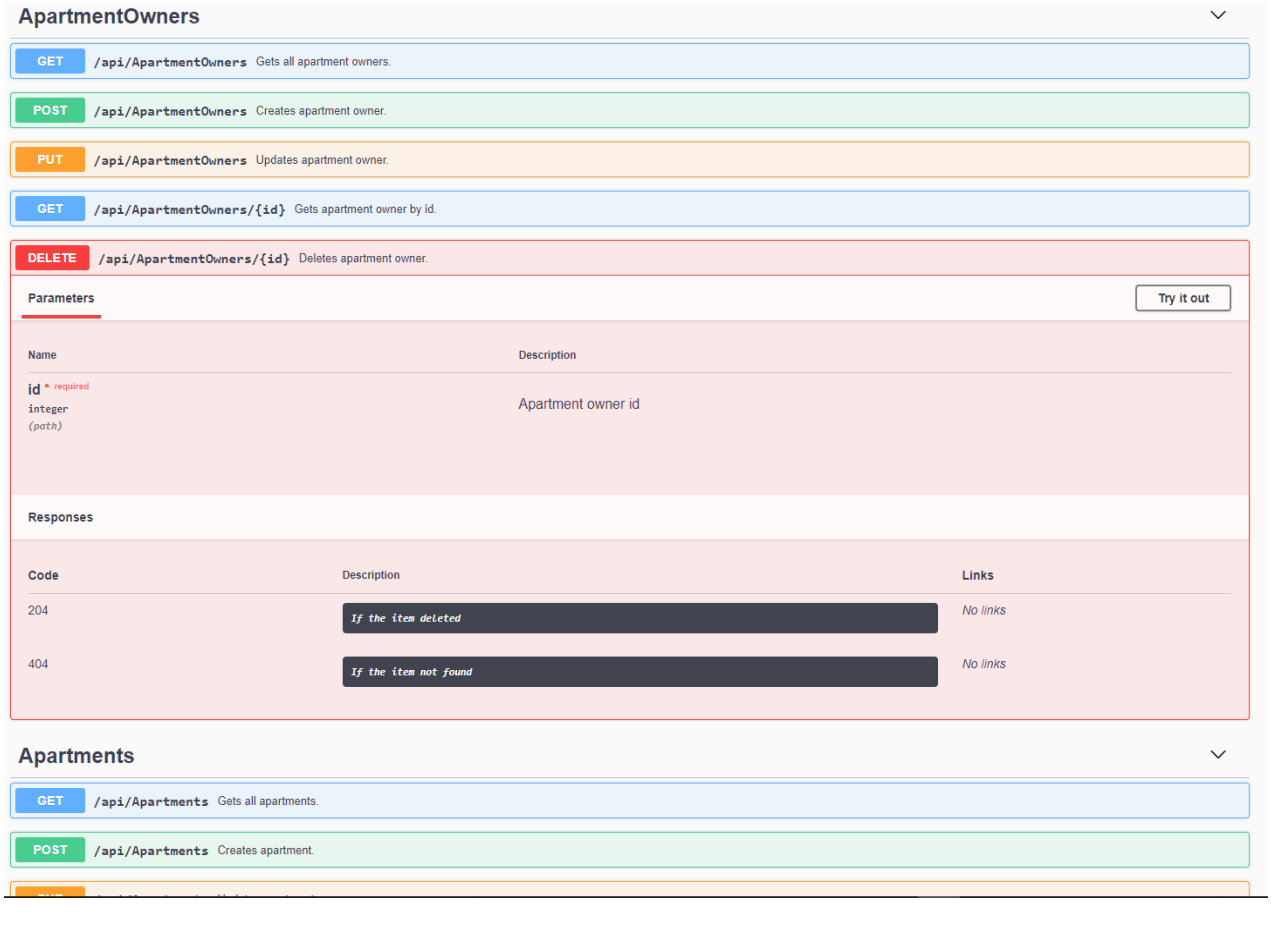

Рисунок 3.21 – Видалення власника

Далі розглянутий мехіназм додавання до своїх активів квартири. Для цього потрібно заповнити певні поля. Такі як: адреса, кількість каімнат, поверх, площа та опис її як такої. Опис потрібен для того щоб показати кращі сторони об'єкту, а

саме – інфраструктуру, технічне обладнання та іншу додаткову інформацію (рис. 3.22):

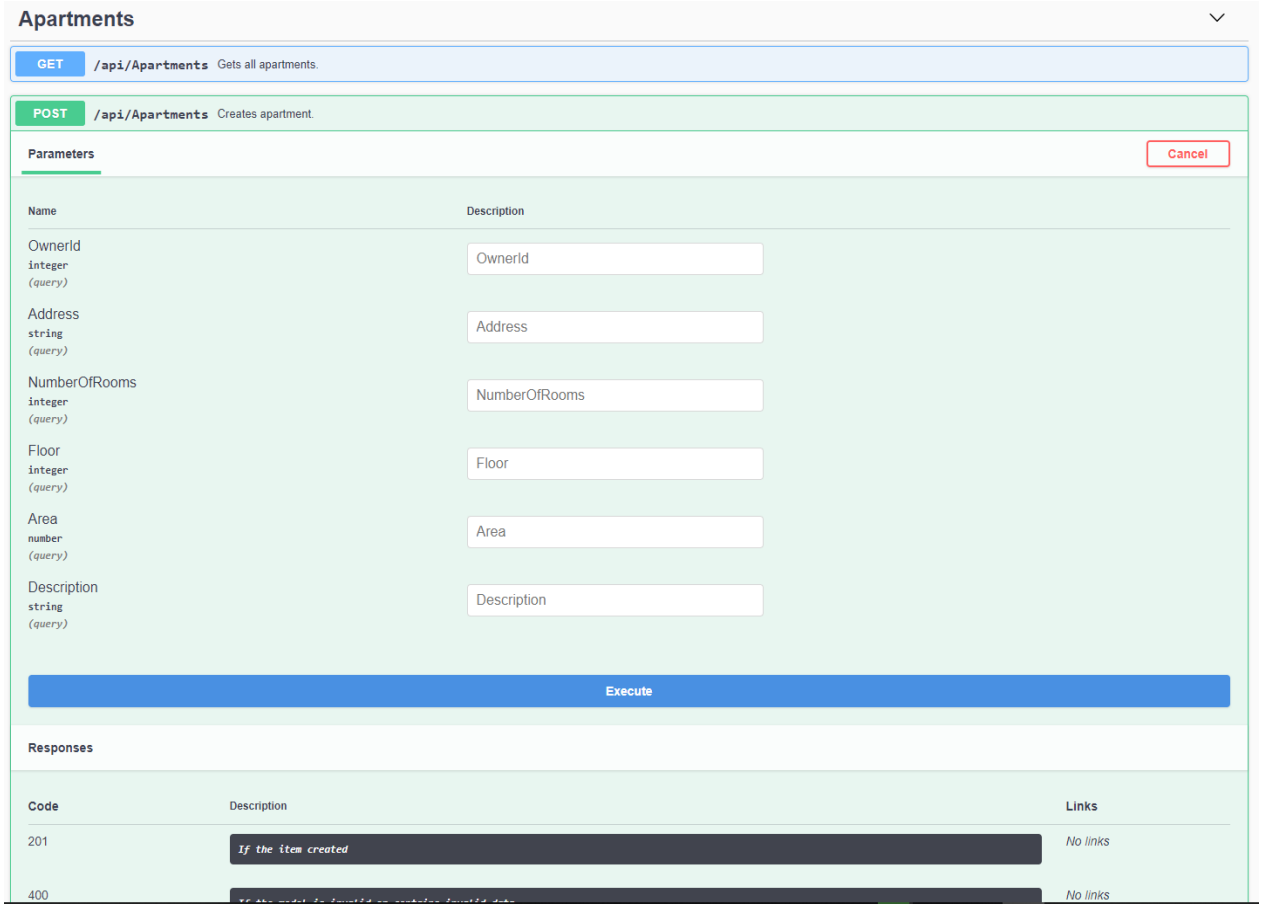

Рисунок 3.22 – Додавання квартири

Після того як у користувача з'явилася квартира він може розміщювати оголошення купівлі/продажу. Для створення такого оголошення користувачу необхідно вказати:

— ApartmentId – унікальний ідентифікатор квартири;

— OwnerId – унікальний ідентифікатор власника;

— Rent – сумма за оренду;

— TermInDays – кількість днів, на які власник готовий сдавати свою квартиру.

Пропоную розглянути пропозицію оренди, приведену на рис. 3.23:

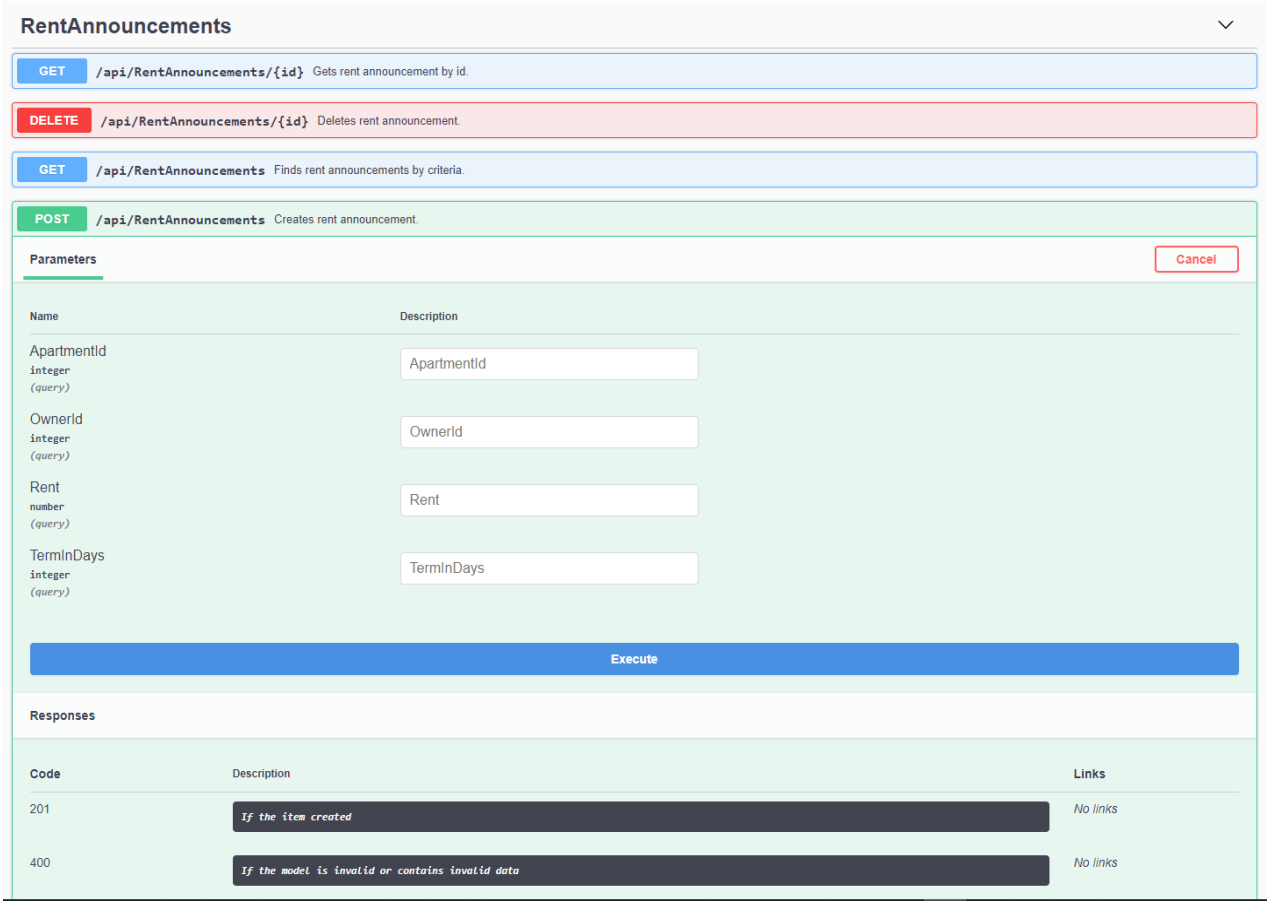

Рисунок 3.23 – Створення оглошення про оренду

3.5 Тестовий приклад роботи бази даних

Побудована база даних основана на реляційній моделі даних, яка складається з сутностей і представлена таким чином, щоб необхідна інформація була знайдена комп'ютером. Вся інформація зберігається у вигляді таблиць. На рис. 3.24, 3.25, 3.26 наведено приклади:

	Id	ApartmentId	ApartmentOwnerId	Status	CreationDate	Discriminator	Rent	TermInDays	Price
1	1	1	1	On review	2018-01-15 00:00:00.0000000	SaleAnnouncement	NULL	NULL	2920000.00
2	2	2	2	On review	2018-02-16 00:00:00.0000000	SaleAnnouncement	NULL	NULL	2050000.00
3	3	3	3	On review	2018-03-17 00:00:00.0000000	SaleAnnouncement	NULL	NULL	1800000.00
4	4	4	4	On review	2018-04-18 00:00:00.0000000	RentAnnouncement	9000.00	180	NULL
5	5	5	5	On review	2018-05-19 00:00:00.0000000	RentAnnouncement	14000.00	365	NULL
6	6	1	1	On review	2019-05-31 20:13:09.3998672	RentAnnouncement	5000.00	10	NULL

Рисунок 3.24 – Таблиці сутності Apartments

	Id	FirstName	LastName	PhoneNumber	Email
1	1	Nikolai	Dzymidzei	88005553535	ne.mogna@gmail.com
2	2	Viktoriya	Rozumskaya	+380976543210	golova68@gmail.com
3	3	Pavel	Katin	+380668882244	assembl9r@gmail.com
4	4	Egor	Nikolaev	+380959116923	goranikonov@gmail.com
5	5	Ivan	Kiryanov	+380971234567	stepan88@gmail.com
6	6	Nikita-Upd	Melnikov-Upd	0665635665	nikmel@gmail.com

Рисунок 3.25– Таблиця сутності ApartmetnOwner

	Id	ApartmentOwnerId	Address	NumberOfRooms	Floor	Area	Description
1	1	1	Kiev, Kudryashova st. 20	2	7	62	Proper repair, with furniture and appliances, air c...
2	2	2	Kiev, Turgenevskaya st. 48	3	3	74,5	Gas column, balcony, needs repair.
3	3	3	Kiev, Zakrevskogo st. 95	3	6	80	Floor heating, fireplace (electric), direct transport ...
4	4	4	Kiev, Vasilkovskaya st. 23/16	2	2	57	Built-in kitchen, stove Bosch, hood Cata. The ba...
5	5	5	Kiev, Timoshenko st. 13a	3	18	85	

Рисунок 3.26– Таблиця сутності Apartments

3.6 Варіанти подальшого розвитку роботи

Оскільки програмний продукт був виконаний з дотриманням принципів SOLID та реалізацією трьохрівневої архітектури, то є можливість імплементувати подальший функціонал в майбутньому. Подальший розвиток роботи складається як з розширення вже реалізованих можливостей, так і додавання нових:

1. Редагування профілю користувача в особистому кабінеті;
2. Завантаження фотографій приміщень в об’яві;
3. Відображення розташування приміщень на мапі;
4. Реалізація front-end частини;
5. Система коментарів та оцінок під оголошеннями або сторінкою профілю власників квартир.

3.7 Висновки до розділу 3

Оцінивши час, наданий на реалізацію дипломного проетку, була проведена декомпозиція завдань та складений план роботи. В результаті отримано програмний продукт, веб-застосунок для оренда та продажу нерузомості.

ВИСНОВКИ

В даній роботі було розроблено веб-застосунок для оренди та продажу нерухомості типу WebApi за допомогою сучасних засобів. Були використані знання по роботі з технологіями ASP.NET, Entity Framework і проектуванні баз даних.

Для виконання поставленого завдання було проаналізовано сучасні можливості ринку і створено більш зручний за функціоналом та швидкодією додаток. Також було вибрано найзручнішу СУБД – SSMS. Вона э безкоштовною і повністю задовольняє технічним вимогам.

Побудовано зручний інтерфейс для користувача. Розроблено трьох-рівневу архітектуру додатку для його кращої функціональності. У ході проектування було реалізовано принципи SOLID та виконано умови C# Code Conventions. Також для розуміння було прокоментовано всі класи верхнього рівня.

У результаті було розроблено програмний комплекс, який дає змогу шукати для власних потреб квартиру чи то для оренди, чи то для купівлі/продажу. Система є доволі простою у використанні і безпечною. Сервіс дає змогу авторизуватися в декількох ролях: адміністратор, користувач і гість, дає змогу відшукати необхідну інформацію для кожного.

У рамках дипломного проетку була розроблена серверна частина веб-застосунку для розміщення оголошень про оренду або продаж нерухомості.

ПЕРЕЛІК ПОСИЛАНЬ

1 Get Started with ASP.NET Web API 2 (C#) // https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api, 25.02.2019.

2 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. – М.: ДМК Пресс; Спб.: Питер, 2017. – 896 с.

3 HTTP // https://developer.mozilla.org/ru/docs/Web/HTTP, 25.02.2019.

4 Веб-технологии для разработчиков // https://developer.mozilla.org/ru/docs/Web, 25.02.2019.

5 Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен, Ф. Джепикс. – М.: ДМК Пресс; Спб.: Питер, 2017. – 1328 с.

6 ASP.NET Documentation // https://docs.microsoft.com/en-us/aspnet/, 25.02.2019.

7 IIS Express Overview // https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview, 25.02.2019.

8 SQL Tutorial // https://www.w3schools.com/sql/, 25.02.2019.

9 Database from Oracle // https://www.oracle.com/database/, 25.02.2019.

10 SQL Server Management Studio (SSMS) // https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017, 25.02.2019.

11 Entity Framework Core // https://docs.microsoft.com/en-us/ef/core/, 26.02.2019.

12 ADO.NET code examples // https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-code-examples, 26.02.2019.

13 Language Integrated Query (LINQ) // https://docs.microsoft.com/en-us/dotnet/csharp/linq/, 26.02.2019.

14 SOLID principles // https://www.dotnetcurry.com/software-gardening/1365/solid-principles, 26.02.2019.

15 .NET Core Guide // https://docs.microsoft.com/en-us/dotnet/core/, 27.02.2019.

16 Dependency injection // https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.2, 27.02.2019.

17 IoC Container // https://www.tutorialsteacher.com/ioc/ioc-container, 27.02.2019.

18 C# Coding Conventions // https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions, 28.02.2019.

19 GC Class // https://docs.microsoft.com/en-us/dotnet/api/system.gc?view=netframework-4.8, 28.02.2019.

20 Three Layer Architecture in C# .NET // https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET-2, 01.03.2019.

21 Presentation Layer // https://www.techopedia.com/definition/8955/presentation-layer, 01.03.2019.

22 Business Logic Layer // https://metanit.com/sharp/mvc5/23.8.php, 02.03.2019.

23 Data access layer // https://en.wikipedia.org/wiki/Data_access_layer, 02.03.2019.

24 Repository // https://martinfowler.com/eaaCatalog/repository.html, 03.03.2019.

25 Unit of Work // https://martinfowler.com/eaaCatalog/unitOfWork.html, 04.03.2019.

26 Data Transfer Objects (DTOs) // https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5, 05.03.2019.

27 Email validation rules // https://help.returnpath.com/hc/en-us/articles/220560587-What-are-the-rules-for-email-address-syntax, 10.03.2019.

28 Password validation rules // https://stackoverflow.com/questions/31539727/laravel-password-validation-rule, 10.03.2019.

29 Response code 200 // https://developer.mozilla.org/ru/docs/Web/HTTP/Status/200, 12.03.2019.

30 Reponse code 400 // https://developer.mozilla.org/ru/docs/Web/HTTP/Status/400, 12.03.2019.

```
namespace EstateAgency.DAL.Entities.Base
{
    public abstract class Entity
    {
        public int Id { get; set; }
    }
}

using System;
using EstateAgency.DAL.Entities.Base;

namespace EstateAgency.DAL.Entities
{
    public class Announcement : Entity
    {
        public Apartment Apartment { get; set; }
        public int ApartmentId { get; set; }
        public ApartmentOwner ApartmentOwner { get; set; }
        public int ApartmentOwnerId { get; set; }
        public string Status { get; set; }
        public DateTime CreationDate { get; set; }
    }
}

using System.Collections.Generic;
using EstateAgency.DAL.Entities.Base;

namespace EstateAgency.DAL.Entities
{
    public class Apartment : Entity
    {
        public ApartmentOwner ApartmentOwner { get; set; }
        public int ApartmentOwnerId { get; set; }
        public ICollection<Announcement> Announcements { get; set; }
        public string Address { get; set; }
        public byte NumberOfRooms { get; set; }
        public byte Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

using System.Collections.Generic;
using EstateAgency.DAL.Entities.Base;

namespace EstateAgency.DAL.Entities
{
    public class ApartmentOwner : Entity
    {
        public ICollection<Announcement> Announcements { get; set; }
        public ICollection<Apartment> Apartments { get; set; }
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

namespace EstateAgency.DAL.Entities
{
    public class RentAnnouncement : Announcement
    {
        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

namespace EstateAgency.DAL.Entities
{
    public class SaleAnnouncement : Announcement
    {
        public decimal Price { get; set; }
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories.Announcements
{
    public class AnnouncementRepository : IAnnouncementRepository
    {
        private readonly DbSet<Announcement> _announcements;
        private readonly ApplicationDbContext _dbContext;
        private bool _disposed = false;

        public AnnouncementRepository(ApplicationDbContext dbContext)
        {
            _dbContext = dbContext;
            _announcements = _dbContext.Announcements;
        }

        public async Task<Announcement> GetAsync(int id)
        {
            return await _announcements
                .Include(a => a.ApartmentOwner)
                .Include(a => a.Apartment)
                .FirstOrDefaultAsync(a => a.Id == id);
        }
```

```
        public IQueryable<Announcement> GetAll()
        {
            return _announcements
                .Include(a => a.ApartmentOwner)
                .Include(a => a.Apartment);
        }

        public void Update(Announcement entity)
        {
            _announcements.Update(entity);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected void Dispose(bool disposing)
        {
            if (_disposed) return;

            if (disposing)
            {
                _dbContext?.Dispose();
            }

            _disposed = true;
        }

        ~AnnouncementRepository()
        {
            Dispose(false);
        }
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories.ApartmentOwners
{
    public class ApartmentOwnerRepository : Repository<ApartmentOwner>,
IApartmentOwnerRepository
    {
        private readonly DbSet<ApartmentOwner> _apartmentOwners;
```

```
        public ApartmentOwnerRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
            _apartmentOwners = DbContext.ApartmentOwners;
        }

        public override IQueryable<ApartmentOwner> GetAll()
        {
            return _apartmentOwners
                .Include(ao => ao.Announcements)
                .Include(ao => ao.Apartments);
        }

        public override async Task<ApartmentOwner> GetAsync(int id)
        {
            return await _apartmentOwners
                .Include(ao => ao.Announcements)
                .Include(ao => ao.Apartments)
                .FirstOrDefaultAsync(ao => ao.Id == id);
        }

        public override async Task AddAsync(ApartmentOwner entity)
        {
            await _apartmentOwners.AddAsync(entity);
        }

        public override void Delete(int id)
        {
            var apartmentOwner = new ApartmentOwner { Id = id };
            _apartmentOwners.Remove(apartmentOwner);
        }

        public override void Update(ApartmentOwner entity)
        {
            _apartmentOwners.Update(entity);
        }

        public override async Task<bool> ContainsEntityWithId(int id)
        {
            return await _apartmentOwners.AnyAsync(ao => ao.Id == id);
        }
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories.Apartments
{
```

```
    public class ApartmentRepository : Repository<Apartment>, IApartmentRepository
    {
        private readonly DbSet<Apartment> _apartments;

        public ApartmentRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
            _apartments = DbContext.Apartments;
        }

        public override IQueryable<Apartment> GetAll()
        {
            return _apartments
                .Include(a => a.ApartmentOwner)
                .Include(a => a.Announcements);
        }

        public override async Task<Apartment> GetAsync(int id)
        {
            return await _apartments
                .Include(a => a.ApartmentOwner)
                .Include(a => a.Announcements)
                .FirstOrDefaultAsync(a => a.Id == id);
        }

        public override async Task AddAsync(Apartment entity)
        {
            await _apartments.AddAsync(entity);
        }

        public override void Delete(int id)
        {
            var apartment = new Apartment { Id = id };
            _apartments.Remove(apartment);
        }

        public override void Update(Apartment entity)
        {
            _apartments.Update(entity);
        }

        public override async Task<bool> ContainsEntityWithId(int id)
        {
            return await _apartments.AnyAsync(a => a.Id == id);
        }
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using System.Linq;
```

```
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories.RentAnnouncements
{
    public class RentAnnouncementRepository : Repository<RentAnnouncement>,
IRentAnnouncementRepository
    {
        private readonly DbSet<RentAnnouncement> _rentAnnouncements;

        public RentAnnouncementRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
            _rentAnnouncements = DbContext.RentAnnouncements;
        }

        public override IQueryable<RentAnnouncement> GetAll()
        {
            return _rentAnnouncements
                .Include(ra => ra.ApartmentOwner)
                .Include(ra => ra.Apartment);
        }

        public override async Task<RentAnnouncement> GetAsync(int id)
        {
            return await _rentAnnouncements
                .Include(ra => ra.ApartmentOwner)
                .Include(ra => ra.Apartment)
                .FirstOrDefaultAsync(ra => ra.Id == id);
        }

        public override async Task AddAsync(RentAnnouncement entity)
        {
            await _rentAnnouncements.AddAsync(entity);
        }

        public override void Delete(int id)
        {
            var rentAnnouncement = new RentAnnouncement { Id = id };
            _rentAnnouncements.Remove(rentAnnouncement);
        }

        public override void Update(RentAnnouncement entity)
        {
            _rentAnnouncements.Update(entity);
        }

        public override async Task<bool> ContainsEntityWithId(int id)
        {
            return await _rentAnnouncements.AnyAsync(ra => ra.Id == id);
        }
    }
}
```

```
using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories.RentAnnouncements
{
    public class RentAnnouncementRepository : Repository<RentAnnouncement>,
IRentAnnouncementRepository
    {
        private readonly DbSet<RentAnnouncement> _rentAnnouncements;

        public RentAnnouncementRepository(ApplicationDbContext applicationDbContext) :
base(applicationDbContext)
        {
            _rentAnnouncements = DbContext.RentAnnouncements;
        }

        public override IQueryable<RentAnnouncement> GetAll()
        {
            return _rentAnnouncements
                .Include(ra => ra.ApartmentOwner)
                .Include(ra => ra.Apartment);
        }

        public override async Task<RentAnnouncement> GetAsync(int id)
        {
            return await _rentAnnouncements
                .Include(ra => ra.ApartmentOwner)
                .Include(ra => ra.Apartment)
                .FirstOrDefaultAsync(ra => ra.Id == id);
        }

        public override async Task AddAsync(RentAnnouncement entity)
        {
            await _rentAnnouncements.AddAsync(entity);
        }

        public override void Delete(int id)
        {
            var rentAnnouncement = new RentAnnouncement { Id = id };
            _rentAnnouncements.Remove(rentAnnouncement);
        }

        public override void Update(RentAnnouncement entity)
        {
            _rentAnnouncements.Update(entity);
        }

        public override async Task<bool> ContainsEntityWithId(int id)
```

```
        {
            return await _rentAnnouncements.AnyAsync(ra => ra.Id == id);
        }
    }
}

using EstateAgency.DAL.Entities.Base;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories
{
    public interface IRepository<TEntity> : IDisposable
        where TEntity : Entity

    {
        /// <summary>
        /// Gets entity by id from data source.
        /// </summary>
        /// <param name="id">Entity id</param>
        /// <returns>Returns entity by id</returns>
        Task<TEntity> GetAsync(int id);

        /// <summary>
        /// Gets all entities from data source.
        /// </summary>
        /// <returns>Returns all entities</returns>
        IQueryable<TEntity> GetAll();

        /// <summary>
        /// Deletes entity by id in data source.
        /// </summary>
        /// <param name="id">Entity id</param>
        void Delete(int id);

        /// <summary>
        /// Adds entity to data source.
        /// </summary>
        /// <param name="entity">Entity to add</param>
        Task AddAsync(TEntity entity);

        /// <summary>
        /// Updates entity in data source.
        /// </summary>
        /// <param name="entity">Entity to update</param>
        void Update(TEntity entity);

        /// <summary>
        /// Checks if entity with id exist in data source.
        /// </summary>
        /// <param name="id">Entity id</param>
```

```
        /// <returns>Returns flag whether entity exists in data source</returns>
        Task<bool> ContainsEntityWithId(int id);
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Entities.Base;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.DAL.Repositories
{
    public abstract class Repository<TEntity> : IRepository<TEntity> where TEntity : Entity
    {
        protected readonly ApplicationDbContext DbContext;
        private bool _disposed = false;

        protected Repository(ApplicationDbContext dbContext)
        {
            DbContext = dbContext;
        }

        public abstract Task<TEntity> GetAsync(int id);
        public abstract IQueryable<TEntity> GetAll();
        public abstract void Delete(int id);
        public abstract Task AddAsync(TEntity entity);
        public abstract void Update(TEntity entity);
        public abstract Task<bool> ContainsEntityWithId(int id);

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected void Dispose(bool disposing)
        {
            if (_disposed) return;

            if (disposing)
            {
                DbContext?.Dispose();
            }

            _disposed = true;
        }

        ~Repository()
        {
            Dispose(false);
        }
```

```
    }
}

using EstateAgency.DAL.Repositories.Announcements;
using EstateAgency.DAL.Repositories.ApartmentOwners;
using EstateAgency.DAL.Repositories.Apartments;
using EstateAgency.DAL.Repositories.RentAnnouncements;
using EstateAgency.DAL.Repositories.SaleAnnouncements;
using System;
using System.Threading.Tasks;

namespace EstateAgency.DAL.UnitOfWork
{
    public interface IUnitOfWork : IDisposable
    {
        IApartmentOwnerRepository ApartmentOwners { get; }
        IApartmentRepository Apartments { get; }
        ISaleAnnouncementRepository SaleAnnouncements { get; }
        IRentAnnouncementRepository RentAnnouncements { get; }
        IAnnouncementRepository Announcements { get; }

        void SaveChanges();
        Task SaveChangesAsync();
    }
}

using EstateAgency.DAL.EF;
using EstateAgency.DAL.Repositories.Announcements;
using EstateAgency.DAL.Repositories.ApartmentOwners;
using EstateAgency.DAL.Repositories.Apartments;
using EstateAgency.DAL.Repositories.RentAnnouncements;
using EstateAgency.DAL.Repositories.SaleAnnouncements;
using System;
using System.Threading.Tasks;

namespace EstateAgency.DAL.UnitOfWork
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly ApplicationDbContext _dbContext;
        private bool _disposed = false;

        public UnitOfWork(ApplicationDbContext context)
        {
            _dbContext = context;
        }

        public IApartmentOwnerRepository ApartmentOwners
        {
            get
            {
                if (_apartmentOwnerRepository == null)
```

```
            {
                _apartmentOwnerRepository = new ApartmentOwnerRepository(_dbContext);
            }
            return _apartmentOwnerRepository;
        }
    }
    private IApartmentOwnerRepository _apartmentOwnerRepository;

    public IApartmentRepository Apartments
    {
        get
        {
            if (_apartmentRepository == null)
            {
                _apartmentRepository = new ApartmentRepository(_dbContext);
            }
            return _apartmentRepository;
        }
    }
    private IApartmentRepository _apartmentRepository;

    public ISaleAnnouncementRepository SaleAnnouncements
    {
        get
        {
            if (_saleAnnouncementRepository == null)
            {
                _saleAnnouncementRepository = new SaleAnnouncementRepository(_dbContext);
            }
            return _saleAnnouncementRepository;
        }
    }
    private ISaleAnnouncementRepository _saleAnnouncementRepository;

    public IRentAnnouncementRepository RentAnnouncements
    {
        get
        {
            if (_rentAnnouncementRepository == null)
            {
                _rentAnnouncementRepository = new RentAnnouncementRepository(_dbContext);
            }
            return _rentAnnouncementRepository;
        }
    }
    private IRentAnnouncementRepository _rentAnnouncementRepository;

    public IAnnouncementRepository Announcements
    {
        get
        {
            if (_announcementRepository == null)
```

```
            {
                _announcementRepository = new AnnouncementRepository(_dbContext);
            }
            return _announcementRepository;
        }
    }
    private IAnnouncementRepository _announcementRepository;

    public void SaveChanges()
    {
        _dbContext.SaveChanges();
    }

    public async Task SaveChangesAsync()
    {
        await _dbContext.SaveChangesAsync();
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected void Dispose(bool disposing)
    {
        if (_disposed) return;

        if (disposing)
        {
            ApartmentOwners?.Dispose();
            Apartments?.Dispose();
            SaleAnnouncements?.Dispose();
            RentAnnouncements?.Dispose();
            Announcements?.Dispose();

            _dbContext?.Dispose();
        }

        _disposed = true;
    }

    ~UnitOfWork()
    {
        Dispose(false);
    }
  }
}

using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;
```

```
namespace EstateAgency.DAL.EF
{
    public class ApplicationDbContext : DbContext
    {
        public DbSet<ApartmentOwner> ApartmentOwners { get; set; }
        public DbSet<Apartment> Apartments { get; set; }
        public DbSet<Announcement> Announcements { get; set; }
        public DbSet<SaleAnnouncement> SaleAnnouncements { get; set; }
        public DbSet<RentAnnouncement> RentAnnouncements { get; set; }

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> dbContextOptions) :
base(dbContextOptions)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Announcement>()
                .HasOne(a => a.Apartment)
                .WithMany(a => a.Announcements)
                .HasForeignKey(a => a.ApartmentId)
                .OnDelete(DeleteBehavior.Restrict);

            DatabaseSeeder.Seed(modelBuilder);
            base.OnModelCreating(modelBuilder);
        }
    }
}

using System;
using EstateAgency.DAL.Entities;
using Microsoft.EntityFrameworkCore;

namespace EstateAgency.DAL.EF
{
    public static class DatabaseSeeder
    {
        public static void Seed(ModelBuilder modelBuilder)
        {
            //ApartmentOwners
            modelBuilder.Entity<ApartmentOwner>().HasData(
                new ApartmentOwner
                {
                    Id = 1,
                    FirstName = "Nikolai",
                    LastName = "Dzymidzei",
                    PhoneNumber = "88005553535",
                    Email = "ne.mogna@gmail.com"
                },
                new ApartmentOwner
                {
```

```
            Id = 2,
            FirstName = "Viktoriya",
            LastName = "Rozumskaya",
            PhoneNumber = "+380976543210",
            Email = "golova68@gmail.com"
        },
        new ApartmentOwner
        {
            Id = 3,
            FirstName = "Pavel",
            LastName = "Katin",
            PhoneNumber = "+380668882244",
            Email = "assembl9r@gmail.com"
        },
        new ApartmentOwner
        {
            Id = 4,
            FirstName = "Egor",
            LastName = "Nikolaev",
            PhoneNumber = "+380959116923",
            Email = "goranikonov@gmail.com"
        },
        new ApartmentOwner
        {
            Id = 5,
            FirstName = "Ivan",
            LastName = "Kiryanov",
            PhoneNumber = "+380971234567",
            Email = "stepan88@gmail.com"
        }
    );

    //Apartments
    modelBuilder.Entity<Apartment>().HasData(
        new Apartment
        {
            Id = 1,
            ApartmentOwnerId = 1,
            Address = "Kiev, Kudryashova st. 20",
            NumberOfRooms = 2,
            Area = 62.0,
            Floor = 7,
            Description =
                "Proper repair, with furniture and appliances, air conditioning in every room," +
                "guarded entrance, near Solomensky Park."
        },
        new Apartment
        {
            Id = 2,
            ApartmentOwnerId = 2,
            Address = "Kiev, Turgenevskaya st. 48",
            NumberOfRooms = 3,
```

```
                Area = 74.5,
                Floor = 3,
                Description = "Gas column, balcony, needs repair."
            },
              new Apartment
            {
                Id = 3,
                ApartmentOwnerId = 3,
                Address = "Kiev, Zakrevskogo st. 95",
                NumberOfRooms = 3,
                Area = 80.0,
                Floor = 6,
                Description = "Floor heating, fireplace (electric), direct transport to the metro."
            },
            new Apartment
            {
                Id = 4,
                ApartmentOwnerId = 4,
                Address = "Kiev, Vasilkovskaya st. 23/16",
                NumberOfRooms = 2,
                Area = 57.0,
                Floor = 2,
                Description =
                    "Built-in kitchen, stove Bosch, hood Cata. The bathroom has an Electrolux boiler." +
                    "The balcony is glazed. Armored door. Easy parking in the yard. Good infrastructure:
metro Vasilkovskaya 300 meters." +
                    "Goloseevsky park 600 meters. Kindergarten near the house. Many supermarkets and
shops"
            },
            new Apartment
            {
                Id = 5,
                ApartmentOwnerId = 5,
                Address = "Kiev, Timoshenko st. 13a",
                NumberOfRooms = 3,
                Area = 85.0,
                Floor = 18,
                Description = ""
            }
        );

        //SaleAnnouncements
        modelBuilder.Entity<SaleAnnouncement>().HasData(
            new SaleAnnouncement
            {
                Id = 1,
                ApartmentOwnerId = 1,
                ApartmentId = 1,
                Price = 2920000,
                Status = "On review",
                CreationDate = new DateTime(2018, 1, 15)
            },
```

```
            new SaleAnnouncement
            {
                Id = 2,
                ApartmentOwnerId = 2,
                ApartmentId = 2,
                Price = 2050000,
                Status = "On review",
                CreationDate = new DateTime(2018, 2, 16)
            },
            new SaleAnnouncement
            {
                Id = 3,
                ApartmentOwnerId = 3,
                ApartmentId = 3,
                Price = 1800000,
                Status = "On review",
                CreationDate = new DateTime(2018, 3, 17)
            }
        );

        //RentAnnouncement
        modelBuilder.Entity<RentAnnouncement>().HasData(
            new RentAnnouncement
            {
                Id = 4,
                ApartmentOwnerId = 4,
                ApartmentId = 4,
                Rent = 9000,
                TermInDays = 180,
                Status = "On review",
                CreationDate = new DateTime(2018, 4, 18)
            },
            new RentAnnouncement
            {
                Id = 5,
                ApartmentOwnerId = 5,
                ApartmentId = 5,
                Rent = 14000,
                TermInDays = 365,
                Status = "On review",
                CreationDate = new DateTime(2018, 5, 19)
            }
        );
    }
  }
}

using AutoMapper;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.Announcements
{
```

```
    public class AnnouncementEntityDtoMappingProfile : Profile
    {
        public AnnouncementEntityDtoMappingProfile()
        {
            CreateMap<Announcement, AnnouncementDto>()
                .ForMember(dst => dst.ApartmentAddress, src => src.MapFrom(ra =>
ra.Apartment.Address))
                .ForMember(dst => dst.ApartmentArea, src => src.MapFrom(ra => ra.Apartment.Area))
                .ForMember(dst => dst.ApartmentDescription, src => src.MapFrom(ra =>
ra.Apartment.Description))
                .ForMember(dst => dst.ApartmentFloor, src => src.MapFrom(ra => ra.Apartment.Floor))
                .ForMember(dst => dst.ApartmentNumberOfRooms, src => src.MapFrom(ra =>
ra.Apartment.NumberOfRooms))
                .ForMember(dst => dst.OwnerEmail, src => src.MapFrom(ra =>
ra.ApartmentOwner.Email))
                .ForMember(dst => dst.OwnerFirstName, src => src.MapFrom(ra =>
ra.ApartmentOwner.FirstName))
                .ForMember(dst => dst.OwnerLastName, src => src.MapFrom(ra =>
ra.ApartmentOwner.LastName))
                .ForMember(dst => dst.OwnerPhoneNumber, src => src.MapFrom(ra =>
ra.ApartmentOwner.PhoneNumber));
        }
    }
}

namespace EstateAgency.BLL.Announcements
{
    public class AnnouncementDto
    {
        public int Id { get; set; }

        public string ApartmentAddress { get; set; }
        public byte ApartmentNumberOfRooms { get; set; }
        public byte ApartmentFloor { get; set; }
        public double ApartmentArea { get; set; }
        public string ApartmentDescription { get; set; }

        public string OwnerFirstName { get; set; }
        public string OwnerLastName { get; set; }
        public string OwnerPhoneNumber { get; set; }
        public string OwnerEmail { get; set; }

        public string Status { get; set; }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;

namespace EstateAgency.BLL.Announcements.Services
{
    public interface IAnnouncementService
```

```
    {
        /// <summary>
        /// Update status of the announcement with given id.
        /// </summary>
        /// <param name="announcementId">Announcement id</param>
        /// <param name="status">New status for updating</param>
        Task UpdateStatusAsync(int announcementId, string status);

        /// <summary>
        /// Gets all announcements with "On review" status.
        /// </summary>
        /// <returns>Returns announcements with "On review" status</returns>
        Task<IEnumerable<AnnouncementDto>> GetAllToReviewAsync();

        /// <summary>
        /// Get announcement by given id.
        /// </summary>
        /// <param name="id">Announcement id</param>
        /// <returns>Returns announcement by given id</returns>
        Task<AnnouncementDto> GetAsync(int id);
    }
}

using AutoMapper;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.BLL.Announcements.Services
{
    public class AnnouncementService : IAnnouncementService
    {
        private readonly IUnitOfWork _estateAgencyDataStorage;
        private readonly IMapper _mapper;

        public AnnouncementService(IUnitOfWork estateAgencyDataStorage, IMapper mapper)
        {
            _estateAgencyDataStorage = estateAgencyDataStorage;
            _mapper = mapper;
        }

        public async Task UpdateStatusAsync(int announcementId, string status)
        {
            var announcementToUpdate = await
_estateAgencyDataStorage.Announcements.GetAsync(announcementId);
            announcementToUpdate.Status = status;
            _estateAgencyDataStorage.Announcements.Update(announcementToUpdate);
            await _estateAgencyDataStorage.SaveChangesAsync();
        }
```

```
        public async Task<IEnumerable<AnnouncementDto>> GetAllToReviewAsync()
        {
            var announcementsToReview =
                await _estateAgencyDataStorage.Announcements.GetAll()
                .Where(a => a.Status == "On review").ToListAsync();
            return _mapper.Map<IEnumerable<AnnouncementDto>>(announcementsToReview);
        }

        public async Task<AnnouncementDto> GetAsync(int id)
        {
            var announcementToReview =
                await _estateAgencyDataStorage.Announcements.GetAsync(id);
            return _mapper.Map<AnnouncementDto>(announcementToReview);
        }
    }
}

namespace EstateAgency.BLL.ApartmentOwners
{
    public class ApartmentOwnerUpdateDto
    {
        public int Id { get; set; }

        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

using AutoMapper;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.ApartmentOwners
{
    public class ApartmentOwnerEntityDtoMappingProfile : Profile
    {
        public ApartmentOwnerEntityDtoMappingProfile()
        {
            CreateMap<ApartmentOwner, ApartmentOwnerDto>();
            CreateMap<ApartmentOwnerDto, ApartmentOwner>();
            CreateMap<ApartmentOwnerCreateDto, ApartmentOwner>();
        }
    }
}

namespace EstateAgency.BLL.ApartmentOwners
{
    public class ApartmentOwnerDto
    {
        public int Id { get; set; }
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

namespace EstateAgency.BLL.ApartmentOwners
{
    public class ApartmentOwnerCreateDto
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;

namespace EstateAgency.BLL.ApartmentOwners.Services
{
    public interface IApartmentOwnerService
    {
        /// <summary>
        /// Gets all apartment owners.
        /// </summary>
        /// <returns>Returns all apartment owners</returns>
        Task<IEnumerable<ApartmentOwnerDto>> GetAllAsync();

        /// <summary>
        /// Gets apartment owner by id.
        /// </summary>
        /// <param name="id">Apartment owner id</param>
        /// <returns>Returns apartment owner by id</returns>
        Task<ApartmentOwnerDto> GetAsync(int id);

        /// <summary>
        /// Creates apartment.
        /// </summary>
        /// <param name="apartmentOwnerCreateDto">Apartment owner creation model</param>
        /// <returns>Returns id of the created entity or status code in case of failure</returns>
        Task<int> AddAsync(ApartmentOwnerCreateDto apartmentOwnerCreateDto);

        /// <summary>
        /// Updates apartment owner.
        /// </summary>
        /// <param name="apartmentOwnerUpdateDto">Apartment owner updating model</param>
        /// <returns>Returns id of the updated entity or status code in case of failure</returns>
        Task<int> UpdateAsync(ApartmentOwnerUpdateDto apartmentOwnerUpdateDto);
```

```
        /// <summary>
        /// Deletes apartment owner.
        /// </summary>
        /// <param name="id">Apartment owner id</param>
        /// <returns>Returns status code of operation result</returns>
        Task<int> DeleteAsync(int id);
    }
}

using AutoMapper;
using EstateAgency.DAL.Entities;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace EstateAgency.BLL.ApartmentOwners.Services
{
    public class ApartmentOwnerService : IApartmentOwnerService
    {
        private readonly IUnitOfWork _estateAgencyDataStorage;
        private readonly IMapper _mapper;

        public ApartmentOwnerService(IUnitOfWork estateAgencyDataStorage, IMapper mapper)
        {
            _estateAgencyDataStorage = estateAgencyDataStorage;
            _mapper = mapper;
        }

        public async Task<IEnumerable<ApartmentOwnerDto>> GetAllAsync()
        {
            var apartmentOwners = await
_estateAgencyDataStorage.ApartmentOwners.GetAll().ToListAsync();
            var apartmentOwnerDtos =
_mapper.Map<IEnumerable<ApartmentOwnerDto>>(apartmentOwners);
            return apartmentOwnerDtos;
        }

        public async Task<ApartmentOwnerDto> GetAsync(int id)
        {
            var apartmentOwner = await _estateAgencyDataStorage.ApartmentOwners.GetAsync(id);
            var apartmentOwnerDto = _mapper.Map<ApartmentOwnerDto>(apartmentOwner);
            return apartmentOwnerDto;
        }

        public async Task<int> AddAsync(ApartmentOwnerCreateDto apartmentOwnerCreateDto)
        {
            if (string.IsNullOrEmpty(apartmentOwnerCreateDto.FirstName))
            {
                return -1;
            }
            if (string.IsNullOrEmpty(apartmentOwnerCreateDto.LastName))
```

```
            {
                return -2;
            }
            if (string.IsNullOrEmpty(apartmentOwnerCreateDto.Email))
            {
                return -3;
            }
            if (string.IsNullOrEmpty(apartmentOwnerCreateDto.PhoneNumber))
            {
                return -4;
            }

            var apartmentOwner = _mapper.Map<ApartmentOwner>(apartmentOwnerCreateDto);

            await _estateAgencyDataStorage.ApartmentOwners.AddAsync(apartmentOwner);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return apartmentOwner.Id;
        }

        public async Task<int> UpdateAsync(ApartmentOwnerUpdateDto apartmentOwnerUpdateDto)
        {
            if (string.IsNullOrEmpty(apartmentOwnerUpdateDto.FirstName))
            {
                return -1;
            }
            if (string.IsNullOrEmpty(apartmentOwnerUpdateDto.LastName))
            {
                return -2;
            }
            if (string.IsNullOrEmpty(apartmentOwnerUpdateDto.Email))
            {
                return -3;
            }
            if (string.IsNullOrEmpty(apartmentOwnerUpdateDto.PhoneNumber))
            {
                return -4;
            }

            var apartmentOwner = await
_estateAgencyDataStorage.ApartmentOwners.GetAsync(apartmentOwnerUpdateDto.Id);
            apartmentOwner.FirstName = apartmentOwnerUpdateDto.FirstName;
            apartmentOwner.LastName = apartmentOwnerUpdateDto.LastName;
            apartmentOwner.Email = apartmentOwnerUpdateDto.Email;
            apartmentOwner.PhoneNumber = apartmentOwnerUpdateDto.PhoneNumber;

            _estateAgencyDataStorage.ApartmentOwners.Update(apartmentOwner);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return apartmentOwner.Id;
        }

        public async Task<int> DeleteAsync(int id)
        {
```

```
            if (!await _estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(id))
            {
                return -5;
            }

            _estateAgencyDataStorage.ApartmentOwners.Delete(id);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return 1;
        }
    }
}

namespace EstateAgency.BLL.Apartments
{
    public class ApartmentUpdateDto
    {
        public int Id { get; set; }

        public int OwnerId { get; set; }

        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

using AutoMapper;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.Apartments
{
    public class ApartmentEntityDtoMappingProfile : Profile
    {
        public ApartmentEntityDtoMappingProfile()
        {
            CreateMap<Apartment, ApartmentDto>()
                .ForMember(dst => dst.OwnerEmail, src => src.MapFrom(a =>
a.ApartmentOwner.Email))
                .ForMember(dst => dst.OwnerFirstName, src => src.MapFrom(a =>
a.ApartmentOwner.FirstName))
                .ForMember(dst => dst.OwnerLastName, src => src.MapFrom(a =>
a.ApartmentOwner.LastName))
                .ForMember(dst => dst.OwnerPhoneNumber, src => src.MapFrom(a =>
a.ApartmentOwner.PhoneNumber));
            CreateMap<ApartmentDto, Apartment>()
                .ForPath(dst => dst.ApartmentOwner.Email, src => src.MapFrom(ad => ad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwner.FirstName, src => src.MapFrom(ad =>
ad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwner.LastName, src => src.MapFrom(ad =>
ad.OwnerLastName))
```

```
            .ForPath(dst => dst.ApartmentOwner.PhoneNumber, src => src.MapFrom(ad =>
ad.OwnerPhoneNumber));
        }
    }
}

namespace EstateAgency.BLL.Apartments
{
    public class ApartmentDto
    {
        public int Id { get; set; }

        public string OwnerFirstName { get; set; }
        public string OwnerLastName { get; set; }
        public string OwnerPhoneNumber { get; set; }
        public string OwnerEmail { get; set; }

        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

namespace EstateAgency.BLL.Apartments
{
    public class ApartmentCreateDto
    {
        public int OwnerId { get; set; }

        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.Apartments.Services
{
    public interface IApartmentCreator
    {
        /// <summary>
        /// Prepares apartment entity to persist to database.
        /// </summary>
        /// <param name="apartmentCreateDto">Apartment creation DTO</param>
        /// <param name="apartmentOwner">Apartment owner from database</param>
        /// <returns>Returns apartment entity</returns>
        Apartment CreateApartmentForAddition(ApartmentCreateDto apartmentCreateDto,
```

```
            ApartmentOwner apartmentOwner);
    }
}

using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.Apartments.Services
{
    public class ApartmentCreator : IApartmentCreator
    {
        public Apartment CreateApartmentForAddition(ApartmentCreateDto apartmentCreateDto,
            ApartmentOwner apartmentOwner)
        {
            var apartment = new Apartment
            {
                ApartmentOwnerId = apartmentOwner.Id,
                ApartmentOwner = apartmentOwner,

                Address = apartmentCreateDto.Address,
                NumberOfRooms = (byte)apartmentCreateDto.NumberOfRooms,
                Floor = (byte)apartmentCreateDto.Floor,
                Area = apartmentCreateDto.Area,
                Description = apartmentCreateDto.Description
            };
            return apartment;
        }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;

namespace EstateAgency.BLL.Apartments.Services
{
    public interface IApartmentService
    {
        /// <summary>
        /// Gets all apartments.
        /// </summary>
        /// <returns>Returns all apartments</returns>
        Task<IEnumerable<ApartmentDto>> GetAllAsync();

        /// <summary>
        /// Gets apartment by id.
        /// </summary>
        /// <param name="id">Apartment id</param>
        /// <returns>Returns apartment by id</returns>
        Task<ApartmentDto> GetAsync(int id);

        /// <summary>
        /// Creates apartment.
        /// </summary>
```

```
        /// <param name="apartmentCreateDto">Apartment creation model</param>
        /// <returns>Returns id of the created entity or status code in case of failure</returns>
        Task<int> AddAsync(ApartmentCreateDto apartmentCreateDto);

        /// <summary>
        /// Updates apartment.
        /// </summary>
        /// <param name="apartmentUpdateDto">Apartment updating model</param>
        /// <returns>Returns id of the updated entity or status code in case of failure</returns>
        Task<int> UpdateAsync(ApartmentUpdateDto apartmentUpdateDto);

        /// <summary>
        /// Deletes apartment.
        /// </summary>
        /// <param name="id">Apartment id</param>
        /// <returns>Returns status code of operation result</returns>
        Task<int> DeleteAsync(int id);
    }
}

using AutoMapper;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace EstateAgency.BLL.Apartments.Services
{
    public class ApartmentService : IApartmentService
    {
        private readonly IUnitOfWork _estateAgencyDataStorage;
        private readonly IMapper _mapper;
        private readonly IApartmentCreator _apartmentCreator;

        public ApartmentService(IUnitOfWork estateAgencyDataStorage, IMapper mapper,
IApartmentCreator apartmentCreator)
        {
            _estateAgencyDataStorage = estateAgencyDataStorage;
            _mapper = mapper;
            _apartmentCreator = apartmentCreator;
        }

        public async Task<IEnumerable<ApartmentDto>> GetAllAsync()
        {
            var apartments = await _estateAgencyDataStorage.Apartments.GetAll().ToListAsync();
            var apartmentDtos = _mapper.Map<IEnumerable<ApartmentDto>>(apartments);
            return apartmentDtos;
        }

        public async Task<ApartmentDto> GetAsync(int id)
        {
            var apartment = await _estateAgencyDataStorage.Apartments.GetAsync(id);
```

```
            var apartmentDto = _mapper.Map<ApartmentDto>(apartment);
            return apartmentDto;
        }

        public async Task<int> AddAsync(ApartmentCreateDto apartmentCreateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(apartmentCreateDto.OwnerId))
            {
                return -1;
            }
            if (apartmentCreateDto.NumberOfRooms <= 0 || apartmentCreateDto.NumberOfRooms >=
100)
            {
                return -2;
            }
            if (apartmentCreateDto.Area <= 0)
            {
                return -3;
            }
            if (apartmentCreateDto.Floor <= 0 || apartmentCreateDto.Floor >= 100)
            {
                return -4;
            }
            if (string.IsNullOrEmpty(apartmentCreateDto.Address))
            {
                return -5;
            }

            var apartmentOwner = await
_estateAgencyDataStorage.ApartmentOwners.GetAsync(apartmentCreateDto.OwnerId);

            var apartment =
                _apartmentCreator.CreateApartmentForAddition(apartmentCreateDto, apartmentOwner);

            await _estateAgencyDataStorage.Apartments.AddAsync(apartment);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return apartment.Id;
        }

        public async Task<int> UpdateAsync(ApartmentUpdateDto apartmentUpdateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(apartmentUpdateDto.OwnerId))
            {
                return -1;
            }
            if (apartmentUpdateDto.NumberOfRooms <= 0 || apartmentUpdateDto.NumberOfRooms >=
100)
            {
                return -2;
            }
```

```
            if (apartmentUpdateDto.Area <= 0)
            {
                return -3;
            }
            if (apartmentUpdateDto.Floor <= 0 || apartmentUpdateDto.Floor >= 100)
            {
                return -4;
            }
            if (string.IsNullOrEmpty(apartmentUpdateDto.Address))
            {
                return -5;
            }

            var apartment = await
_estateAgencyDataStorage.Apartments.GetAsync(apartmentUpdateDto.Id);
            apartment.NumberOfRooms = (byte)apartmentUpdateDto.NumberOfRooms;
            apartment.Address = apartmentUpdateDto.Address;
            apartment.Floor = (byte)apartmentUpdateDto.Floor;
            apartment.Area = apartmentUpdateDto.Area;
            apartment.Description = apartmentUpdateDto.Description;

            _estateAgencyDataStorage.Apartments.Update(apartment);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return apartment.Id;
        }

        public async Task<int> DeleteAsync(int id)
        {
            if (!await _estateAgencyDataStorage.Apartments.ContainsEntityWithId(id))
            {
                return -6;
            }

            _estateAgencyDataStorage.Apartments.Delete(id);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return 1;
        }
    }
}

using System;

namespace EstateAgency.BLL.RentAnnouncements
{
    public class RentAnnouncementUpdateDto
    {
        public int Id { get; set; }

        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public DateTime CreationDate { get; set; }
```

```
        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

using AutoMapper;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.RentAnnouncements
{
    public class RentAnnouncementEntityDtoMappingProfile : Profile
    {
        public RentAnnouncementEntityDtoMappingProfile()
        {
            CreateMap<RentAnnouncement, RentAnnouncementDto>()
                .ForMember(dst => dst.ApartmentAddress, src => src.MapFrom(ra =>
ra.Apartment.Address))
                .ForMember(dst => dst.ApartmentArea, src => src.MapFrom(ra => ra.Apartment.Area))
                .ForMember(dst => dst.ApartmentDescription, src => src.MapFrom(ra =>
ra.Apartment.Description))
                .ForMember(dst => dst.ApartmentFloor, src => src.MapFrom(ra => ra.Apartment.Floor))
                .ForMember(dst => dst.ApartmentNumberOfRooms, src => src.MapFrom(ra =>
ra.Apartment.NumberOfRooms))
                .ForMember(dst => dst.OwnerEmail, src => src.MapFrom(ra =>
ra.ApartmentOwner.Email))
                .ForMember(dst => dst.OwnerFirstName, src => src.MapFrom(ra =>
ra.ApartmentOwner.FirstName))
                .ForMember(dst => dst.OwnerLastName, src => src.MapFrom(ra =>
ra.ApartmentOwner.LastName))
                .ForMember(dst => dst.OwnerPhoneNumber, src => src.MapFrom(ra =>
ra.ApartmentOwner.PhoneNumber));

            CreateMap<RentAnnouncementDto, RentAnnouncement>()
                .ForPath(dst => dst.Apartment.Address, src => src.MapFrom(rad =>
rad.ApartmentAddress))
                .ForPath(dst => dst.Apartment.Area, src => src.MapFrom(rad => rad.ApartmentArea))
                .ForPath(dst => dst.Apartment.Description, src => src.MapFrom(rad =>
rad.ApartmentDescription))
                .ForPath(dst => dst.Apartment.Floor, src => src.MapFrom(rad => rad.ApartmentFloor))
                .ForPath(dst => dst.Apartment.NumberOfRooms, src => src.MapFrom(rad =>
rad.ApartmentNumberOfRooms))
                .ForPath(dst => dst.ApartmentOwner.Email, src => src.MapFrom(rad =>
rad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwner.FirstName, src => src.MapFrom(rad =>
rad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwner.LastName, src => src.MapFrom(rad =>
rad.OwnerLastName))
                .ForPath(dst => dst.ApartmentOwner.PhoneNumber, src => src.MapFrom(rad =>
rad.OwnerPhoneNumber));
        }
    }
}
```

```
using System;

namespace EstateAgency.BLL.RentAnnouncements
{
    public class RentAnnouncementDto
    {
        public int Id { get; set; }

        public string ApartmentAddress { get; set; }
        public byte ApartmentNumberOfRooms { get; set; }
        public byte ApartmentFloor { get; set; }
        public double ApartmentArea { get; set; }
        public string ApartmentDescription { get; set; }

        public string OwnerFirstName { get; set; }
        public string OwnerLastName { get; set; }
        public string OwnerPhoneNumber { get; set; }
        public string OwnerEmail { get; set; }

        public DateTime CreationDate { get; set; }
        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

using System;

namespace EstateAgency.BLL.RentAnnouncements
{
    public class RentAnnouncementCreateDto
    {
        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public DateTime CreationDate { get; set; }
        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

using System;
using System.Linq.Expressions;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public interface IRentAnnouncementExpressionComposer
    {
        /// <summary>
        /// Composes expression by given parametrs.
        /// </summary>
        /// <param name="maxPrice">Max price in rent announcement</param>
```

```
        /// <param name="numberOfRooms">Number of rooms in rent announcement</param>
        /// <param name="adress">Adress in rent announcement</param>
        /// <returns>Returns expression by given parametrs</returns>
        Expression<Func<RentAnnouncementDto, bool>> Compose(int? maxPrice, int?
numberOfRooms, string adress);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public class RentAnnouncementExpressionComposer : IRentAnnouncementExpressionComposer
    {
        public Expression<Func<RentAnnouncementDto, bool>> Compose(int? maxPrice, int?
numberOfRooms, string adress)
        {
            var expressions = new List<Expression<Func<RentAnnouncementDto, bool>>>();
            expressions.Add(rad => true);

            if (maxPrice.HasValue)
              expressions.Add(rad => rad.Rent < maxPrice.Value);

            if (numberOfRooms.HasValue)
                expressions.Add(rad => numberOfRooms.Value == rad.ApartmentNumberOfRooms);

            if (!string.IsNullOrEmpty(adress))
                expressions.Add(rad => rad.ApartmentAddress.Contains(adress));

            var lambda = GetExpression(expressions.ToArray());
            return lambda;
        }

        private Expression<Func<RentAnnouncementDto, bool>> GetExpression(params
Expression<Func<RentAnnouncementDto, bool>>[] expressions)
        {
            Expression<Func<RentAnnouncementDto, bool>> lambda = expressions[0];
            foreach (var expression in expressions.Skip(1))
            {
                lambda = AndAlso(lambda, expression);
            }
            return lambda;
        }

        private Expression<Func<T, bool>> AndAlso<T>(Expression<Func<T, bool>> expr1,
Expression<Func<T, bool>> expr2)
        {
            var parameter = Expression.Parameter(typeof(T));
```

```
            var leftVisitor = new ReplaceExpressionVisitor(expr1.Parameters[0], parameter);
            var left = leftVisitor.Visit(expr1.Body);

            var rightVisitor = new ReplaceExpressionVisitor(expr2.Parameters[0], parameter);
            var right = rightVisitor.Visit(expr2.Body);

            return Expression.Lambda<Func<T, bool>>(Expression.AndAlso(left, right), parameter);
        }

        private class ReplaceExpressionVisitor
            : ExpressionVisitor
        {
            private readonly Expression _oldValue;
            private readonly Expression _newValue;

            public ReplaceExpressionVisitor(Expression oldValue, Expression newValue)
            {
                _oldValue = oldValue;
                _newValue = newValue;
            }

            public override Expression Visit(Expression node)
            {
                if (node == _oldValue)
                    return _newValue;
                return base.Visit(node);
            }
        }
    }
}

using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public interface IRentAnnouncementCreator
    {
        /// <summary>
        /// Prepares rent announcement entity to persist to database.
        /// </summary>
        /// <param name="rentAnnouncementCreateDto">Rent announcement creation DTO</param>
        /// <param name="apartmentOwner">Apartment owner from database</param>
        /// <param name="apartment">Apartment from database</param>
        /// <returns>Returns rent announcement entity</returns>
        RentAnnouncement CreateRentAnnouncementForAddition(
            RentAnnouncementCreateDto rentAnnouncementCreateDto,
            ApartmentOwner apartmentOwner,
            Apartment apartment);
    }
}

using EstateAgency.DAL.Entities;
```

```
using System;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public class RentAnnouncementCreator : IRentAnnouncementCreator
    {
        public RentAnnouncement
CreateRentAnnouncementForAddition(RentAnnouncementCreateDto rentAnnouncementCreateDto,
            ApartmentOwner apartmentOwner,
            Apartment apartment)
        {
            var rentAnnouncement = new RentAnnouncement
            {
                ApartmentOwnerId = apartmentOwner.Id,
                ApartmentOwner = apartmentOwner,

                ApartmentId = apartment.Id,
                Apartment = apartment,

                CreationDate = DateTime.Now,
                Status = "On review",
                Rent = rentAnnouncementCreateDto.Rent,
                TermInDays = rentAnnouncementCreateDto.TermInDays
            };
            return rentAnnouncement;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public interface IRentAnnouncementService
    {
        /// <summary>
        /// Gets all rent announcements.
        /// </summary>
        /// <returns>Returns all rent announcements</returns>
        Task<IEnumerable<RentAnnouncementDto>> GetAllAsync();

        /// <summary>
        /// Gets rent announcement by id.
        /// </summary>
        /// <param name="id">Rent announcement id</param>
        /// <returns>Returns rent announcement by id</returns>
        Task<RentAnnouncementDto> GetAsync(int id);

        /// <summary>
```

```
        /// Filters rent announcements by expression (filtering criteria).
        /// </summary>
        /// <param name="expression">Expression (filtering criteria)</param>
        /// <returns>Returns rent announcements by expression (filtering criteria)</returns>
        Task<IEnumerable<RentAnnouncementDto>>
FindAsync(Expression<Func<RentAnnouncementDto, bool>> expression);

        /// <summary>
        /// Creates rent announcement.
        /// </summary>
        /// <param name="rentAnnouncementCreateDto">Rent announcement creation model</param>
        /// <returns>Returns id of the created entity or status code in case of failure</returns>
        Task<int> AddAsync(RentAnnouncementCreateDto rentAnnouncementCreateDto);

        /// <summary>
        /// Updates rent announcement.
        /// </summary>
        /// <param name="rentAnnouncementUpdateDto">Rent announcement updating
model</param>
        /// <returns>Returns id of the updated entity or status code in case of failure</returns>
        Task<int> UpdateAsync(RentAnnouncementUpdateDto rentAnnouncementUpdateDto);

        /// <summary>
        /// Deletes rent announcement.
        /// </summary>
        /// <param name="id">Rent announcement id</param>
        /// <returns>Returns status code of operation result</returns>
        Task<int> DeleteAsync(int id);
    }
}

using AutoMapper;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace EstateAgency.BLL.RentAnnouncements.Services
{
    public class RentAnnouncementService : IRentAnnouncementService
    {
        private readonly IUnitOfWork _estateAgencyDataStorage;
        private readonly IMapper _mapper;
        private readonly IRentAnnouncementCreator _rentAnnouncementCreator;

        public RentAnnouncementService(IUnitOfWork estateAgencyDataStorage, IMapper mapper,
IRentAnnouncementCreator rentAnnouncementCreator)
        {
            _estateAgencyDataStorage = estateAgencyDataStorage;
```

```
            _mapper = mapper;
            _rentAnnouncementCreator = rentAnnouncementCreator;
        }

         public async Task<IEnumerable<RentAnnouncementDto>> GetAllAsync()
        {
            var rentAnnouncements = await
_estateAgencyDataStorage.RentAnnouncements.GetAll().ToListAsync();
            var rentAnnouncementDtos =
_mapper.Map<IEnumerable<RentAnnouncementDto>>(rentAnnouncements);
            return rentAnnouncementDtos;
        }

        public async Task<RentAnnouncementDto> GetAsync(int id)
        {
            var rentAnnouncement = await
_estateAgencyDataStorage.RentAnnouncements.GetAsync(id);
            var rentAnnouncementDto = _mapper.Map<RentAnnouncementDto>(rentAnnouncement);
            return rentAnnouncementDto;
        }

        public async Task<IEnumerable<RentAnnouncementDto>>
FindAsync(Expression<Func<RentAnnouncementDto, bool>> expression)
        {
            var rentAnnouncementsQuery = _estateAgencyDataStorage.RentAnnouncements.GetAll();
            var rentAnnouncementDtosQuery =
_mapper.ProjectTo<RentAnnouncementDto>(rentAnnouncementsQuery).Where(expression);
            var rentAnnouncementDtos = await rentAnnouncementDtosQuery.ToListAsync();
            return rentAnnouncementDtos;
        }

        public async Task<int> AddAsync(RentAnnouncementCreateDto rentAnnouncementCreateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(rentAnnouncementCreateDto.O
wnerId))
            {
                return -1;
            }
            if (!await
_estateAgencyDataStorage.Apartments.ContainsEntityWithId(rentAnnouncementCreateDto.Apartme
ntId))
            {
                return -2;
            }
            if (rentAnnouncementCreateDto.Rent <= 0)
            {
                return -3;
            }
            if (rentAnnouncementCreateDto.TermInDays <= 0)
            {
                return -4;
```

```
            }

            var apartmentOwner = await
_estateAgencyDataStorage.ApartmentOwners.GetAsync(rentAnnouncementCreateDto.OwnerId);
            var apartment =
                await
_estateAgencyDataStorage.Apartments.GetAsync(rentAnnouncementCreateDto.ApartmentId);

            var rentAnnouncement =

_rentAnnouncementCreator.CreateRentAnnouncementForAddition(rentAnnouncementCreateDto,
apartmentOwner, apartment);

            rentAnnouncementCreateDto.CreationDate = rentAnnouncement.CreationDate;

            await _estateAgencyDataStorage.RentAnnouncements.AddAsync(rentAnnouncement);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return rentAnnouncement.Id;
        }

        public async Task<int> UpdateAsync(RentAnnouncementUpdateDto
rentAnnouncementUpdateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(rentAnnouncementUpdateDto.
OwnerId))
            {
                return -1;
            }
            if (!await
_estateAgencyDataStorage.Apartments.ContainsEntityWithId(rentAnnouncementUpdateDto.Apartm
entId))
            {
                return -2;
            }
            if (rentAnnouncementUpdateDto.Rent <= 0)
            {
                return -3;
            }
            if (rentAnnouncementUpdateDto.TermInDays <= 0)
            {
                return -4;
            }

            var rentAnnouncement = await
_estateAgencyDataStorage.RentAnnouncements.GetAsync(rentAnnouncementUpdateDto.Id);
            rentAnnouncement.Rent = rentAnnouncementUpdateDto.Rent;
            rentAnnouncement.TermInDays = rentAnnouncementUpdateDto.TermInDays;
            rentAnnouncement.Status = "On review";

            _estateAgencyDataStorage.RentAnnouncements.Update(rentAnnouncement);
            await _estateAgencyDataStorage.SaveChangesAsync();
```

```
            return rentAnnouncement.Id;
        }

        public async Task<int> DeleteAsync(int id)
        {
            if (!await _estateAgencyDataStorage.RentAnnouncements.ContainsEntityWithId(id))
            {
                return -5;
            }

            _estateAgencyDataStorage.RentAnnouncements.Delete(id);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return 1;
        }
    }
}

using System;

namespace EstateAgency.BLL.SaleAnnouncements
{
    public class SaleAnnouncementUpdateDto
    {
        public int Id { get; set; }

        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public DateTime CreationDate { get; set; }
        public decimal Price { get; set; }
    }
}

using AutoMapper;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.SaleAnnouncements
{
    public class SaleAnnouncementEntityDtoMappingProfile : Profile
    {
        public SaleAnnouncementEntityDtoMappingProfile()
        {
            CreateMap<SaleAnnouncement, SaleAnnouncementDto>()
                .ForMember(dst => dst.ApartmentAddress, src => src.MapFrom(sa =>
sa.Apartment.Address))
                .ForMember(dst => dst.ApartmentArea, src => src.MapFrom(sa => sa.Apartment.Area))
                .ForMember(dst => dst.ApartmentDescription, src => src.MapFrom(sa =>
sa.Apartment.Description))
                .ForMember(dst => dst.ApartmentFloor, src => src.MapFrom(sa => sa.Apartment.Floor))
                .ForMember(dst => dst.ApartmentNumberOfRooms, src => src.MapFrom(sa =>
sa.Apartment.NumberOfRooms))
```

```
            .ForMember(dst => dst.OwnerEmail, src => src.MapFrom(sa =>
sa.ApartmentOwner.Email))
            .ForMember(dst => dst.OwnerFirstName, src => src.MapFrom(sa =>
sa.ApartmentOwner.FirstName))
            .ForMember(dst => dst.OwnerLastName, src => src.MapFrom(sa =>
sa.ApartmentOwner.LastName))
            .ForMember(dst => dst.OwnerPhoneNumber, src => src.MapFrom(sa =>
sa.ApartmentOwner.PhoneNumber));
          CreateMap<SaleAnnouncementDto, SaleAnnouncement>()
            .ForPath(dst => dst.Apartment.Address, src => src.MapFrom(sad =>
sad.ApartmentAddress))
            .ForPath(dst => dst.Apartment.Area, src => src.MapFrom(sad => sad.ApartmentArea))
            .ForPath(dst => dst.Apartment.Description, src => src.MapFrom(sad =>
sad.ApartmentDescription))
            .ForPath(dst => dst.Apartment.Floor, src => src.MapFrom(sad => sad.ApartmentFloor))
            .ForPath(dst => dst.Apartment.NumberOfRooms, src => src.MapFrom(sad =>
sad.ApartmentNumberOfRooms))
            .ForPath(dst => dst.ApartmentOwner.Email, src => src.MapFrom(sad =>
sad.OwnerEmail))
            .ForPath(dst => dst.ApartmentOwner.FirstName, src => src.MapFrom(sad =>
sad.OwnerFirstName))
            .ForPath(dst => dst.ApartmentOwner.LastName, src => src.MapFrom(sad =>
sad.OwnerLastName))
            .ForPath(dst => dst.ApartmentOwner.PhoneNumber, src => src.MapFrom(sad =>
sad.OwnerPhoneNumber));
        }
    }
}

using System;

namespace EstateAgency.BLL.SaleAnnouncements
{
    public class SaleAnnouncementDto
    {
        public int Id { get; set; }

        public string ApartmentAddress { get; set; }
        public byte ApartmentNumberOfRooms { get; set; }
        public byte ApartmentFloor { get; set; }
        public double ApartmentArea { get; set; }
        public string ApartmentDescription { get; set; }

        public string OwnerFirstName { get; set; }
        public string OwnerLastName { get; set; }
        public string OwnerPhoneNumber { get; set; }
        public string OwnerEmail { get; set; }

        public DateTime CreationDate { get; set; }
        public decimal Price { get; set; }
    }
}
```

```
using System;

namespace EstateAgency.BLL.SaleAnnouncements
{
    public class SaleAnnouncementCreateDto
    {
        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public DateTime CreationDate { get; set; }
        public decimal Price { get; set; }
    }
}

using System;
using System.Linq.Expressions;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
    public interface ISaleAnnouncementExpressionComposer
    {
        /// <summary>
        /// Composes expression by given parametrs.
        /// </summary>
        /// <param name="maxPrice">Max price in sale announcement</param>
        /// <param name="numberOfRooms">Number of rooms in sale announcement</param>
        /// <param name="adress">Adress in sale announcement</param>
        /// <returns>Returns expression by given parametrs</returns>
        Expression<Func<SaleAnnouncementDto, bool>> Compose(int? maxPrice, int?
numberOfRooms, string adress);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
    public class SaleAnnouncementExpressionComposer : ISaleAnnouncementExpressionComposer
    {
        public Expression<Func<SaleAnnouncementDto, bool>> Compose(int? maxPrice, int?
numberOfRooms, string adress)
        {
            var expressions = new List<Expression<Func<SaleAnnouncementDto, bool>>>();
            expressions.Add(sad => true);

            if (maxPrice.HasValue)
                expressions.Add(sad => sad.Price < maxPrice.Value);
```

```
            if (numberOfRooms.HasValue)
                expressions.Add(sad => numberOfRooms.Value == sad.ApartmentNumberOfRooms);

            if (!string.IsNullOrEmpty(adress))
                expressions.Add(sad => sad.ApartmentAddress.Contains(adress));

            var lambda = GetExpression(expressions.ToArray());
            return lambda;
        }

        private Expression<Func<SaleAnnouncementDto, bool>> GetExpression(params
Expression<Func<SaleAnnouncementDto, bool>>[] expressions)
        {
            Expression<Func<SaleAnnouncementDto, bool>> lambda = expressions[0];
            foreach (var expression in expressions.Skip(1))
            {
                lambda = AndAlso(lambda, expression);
            }
            return lambda;
        }

        private Expression<Func<T, bool>> AndAlso<T>(Expression<Func<T, bool>> expr1,
Expression<Func<T, bool>> expr2)
        {
            var parameter = Expression.Parameter(typeof(T));

            var leftVisitor = new ReplaceExpressionVisitor(expr1.Parameters[0], parameter);
            var left = leftVisitor.Visit(expr1.Body);

            var rightVisitor = new ReplaceExpressionVisitor(expr2.Parameters[0], parameter);
            var right = rightVisitor.Visit(expr2.Body);

            return Expression.Lambda<Func<T, bool>>(Expression.AndAlso(left, right), parameter);
        }

        private class ReplaceExpressionVisitor
            : ExpressionVisitor
        {
            private readonly Expression _oldValue;
            private readonly Expression _newValue;

            public ReplaceExpressionVisitor(Expression oldValue, Expression newValue)
            {
                _oldValue = oldValue;
                _newValue = newValue;
            }

            public override Expression Visit(Expression node)
            {
                if (node == _oldValue)
                    return _newValue;
                return base.Visit(node);
```

```
        }
      }
   }
}

using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
   public interface ISaleAnnouncementCreator
   {
      /// <summary>
      /// Prepares sale announcement entity to persist to database.
      /// </summary>
      /// <param name="saleAnnouncementCreateDto">Sale announcement creation DTO</param>
      /// <param name="apartmentOwner">Apartment owner from database</param>
      /// <param name="apartment">Apartment from database</param>
      /// <returns>Returns sale announcement entity</returns>
      SaleAnnouncement CreateSaleAnnouncementForAddition(
         SaleAnnouncementCreateDto saleAnnouncementCreateDto,
         ApartmentOwner apartmentOwner,
         Apartment apartment);
   }
}

using System;
using EstateAgency.DAL.Entities;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
   public class SaleAnnouncementCreator : ISaleAnnouncementCreator
   {
      public SaleAnnouncement CreateSaleAnnouncementForAddition(SaleAnnouncementCreateDto
saleAnnouncementCreateDto,
         ApartmentOwner apartmentOwner, Apartment apartment)
      {
         var saleAnnouncement = new SaleAnnouncement
         {
            ApartmentOwnerId = apartmentOwner.Id,
            ApartmentOwner = apartmentOwner,

            ApartmentId = apartment.Id,
            Apartment = apartment,

            CreationDate = DateTime.Now,
            Status = "On review",
            Price = saleAnnouncementCreateDto.Price
         };
         return saleAnnouncement;
      }
   }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
    public interface ISaleAnnouncementService
    {
        /// <summary>
        /// Gets all sale announcements.
        /// </summary>
        /// <returns>Returns all sale announcements</returns>
        Task<IEnumerable<SaleAnnouncementDto>> GetAllAsync();

        /// <summary>
        /// Gets sale announcement by id.
        /// </summary>
        /// <param name="id">Sale announcement id</param>
        /// <returns>Returns sale announcement by id</returns>
        Task<SaleAnnouncementDto> GetAsync(int id);

        /// <summary>
        /// Filters sale announcements by expression (filtering criteria).
        /// </summary>
        /// <param name="expression">Expression (filtering criteria)</param>
        /// <returns>Returns sale announcements by expression (filtering criteria)</returns>
        Task<IEnumerable<SaleAnnouncementDto>>
FindAsync(Expression<Func<SaleAnnouncementDto, bool>> expression);

        /// <summary>
        /// Creates sale announcement.
        /// </summary>
        /// <param name="saleAnnouncementCreateDto">Sale announcement creation model</param>
        /// <returns>Returns id of the created entity or status code in case of failure</returns>
        Task<int> AddAsync(SaleAnnouncementCreateDto saleAnnouncementCreateDto);

        /// <summary>
        /// Updates sale announcement.
        /// </summary>
        /// <param name="saleAnnouncementUpdateDto">Sale announcement updating
model</param>
        /// <returns>Returns id of the updated entity or status code in case of failure</returns>
        Task<int> UpdateAsync(SaleAnnouncementUpdateDto saleAnnouncementUpdateDto);

        /// <summary>
        /// Deletes sale announcement.
        /// </summary>
        /// <param name="id">Sale announcement id</param>
        /// <returns>Returns status code of operation result</returns>
        Task<int> DeleteAsync(int id);
```

```
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using AutoMapper;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.EntityFrameworkCore;

namespace EstateAgency.BLL.SaleAnnouncements.Services
{
    public class SaleAnnouncementService : ISaleAnnouncementService
    {
        private readonly IUnitOfWork _estateAgencyDataStorage;
        private readonly IMapper _mapper;
        private readonly ISaleAnnouncementCreator _saleAnnouncementCreator;

        public SaleAnnouncementService(IUnitOfWork estateAgencyDataStorage, IMapper mapper,
ISaleAnnouncementCreator saleAnnouncementCreator)
        {
            _estateAgencyDataStorage = estateAgencyDataStorage;
            _mapper = mapper;
            _saleAnnouncementCreator = saleAnnouncementCreator;
        }

        public async Task<IEnumerable<SaleAnnouncementDto>> GetAllAsync()
        {
            var saleAnnouncements = await
_estateAgencyDataStorage.SaleAnnouncements.GetAll().ToListAsync();
            var saleAnnouncementDtos =
_mapper.Map<IEnumerable<SaleAnnouncementDto>>(saleAnnouncements);
            return saleAnnouncementDtos;
        }

        public async Task<SaleAnnouncementDto> GetAsync(int id)
        {
            var saleAnnouncement = await
_estateAgencyDataStorage.SaleAnnouncements.GetAsync(id);
            var saleAnnouncementDto = _mapper.Map<SaleAnnouncementDto>(saleAnnouncement);
            return saleAnnouncementDto;
        }

        public async Task<IEnumerable<SaleAnnouncementDto>>
FindAsync(Expression<Func<SaleAnnouncementDto, bool>> expression)
        {
            var saleAnnouncementsQuery = _estateAgencyDataStorage.SaleAnnouncements.GetAll();
            var saleAnnouncementDtosQuery =
_mapper.ProjectTo<SaleAnnouncementDto>(saleAnnouncementsQuery).Where(expression);
            var saleAnnouncementDtos = await saleAnnouncementDtosQuery.ToListAsync();
```

```
            return saleAnnouncementDtos;
        }

        public async Task<int> AddAsync(SaleAnnouncementCreateDto saleAnnouncementCreateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(saleAnnouncementCreateDto.O
wnerId))
            {
                return -1;
            }
if (!await
_estateAgencyDataStorage.Apartments.ContainsEntityWithId(saleAnnouncementCreateDto.Apartme
ntId))
            {
                return -2;
            }
            if (saleAnnouncementCreateDto.Price <= 0)
            {
                return -3;
            }

            var apartmentOwner = await
_estateAgencyDataStorage.ApartmentOwners.GetAsync(saleAnnouncementCreateDto.OwnerId);
            var apartment =
                await
_estateAgencyDataStorage.Apartments.GetAsync(saleAnnouncementCreateDto.ApartmentId);

            var saleAnnouncement =

_saleAnnouncementCreator.CreateSaleAnnouncementForAddition(saleAnnouncementCreateDto,
apartmentOwner, apartment);

            saleAnnouncementCreateDto.CreationDate = saleAnnouncement.CreationDate;

            await _estateAgencyDataStorage.SaleAnnouncements.AddAsync(saleAnnouncement);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return saleAnnouncement.Id;
        }

        public async Task<int> UpdateAsync(SaleAnnouncementUpdateDto
saleAnnouncementUpdateDto)
        {
            if (!await
_estateAgencyDataStorage.ApartmentOwners.ContainsEntityWithId(saleAnnouncementUpdateDto.
OwnerId))
            {
                return -1;
            }
            if (!await
_estateAgencyDataStorage.Apartments.ContainsEntityWithId(saleAnnouncementUpdateDto.Apartm
entId))
```

```
            {
                return -2;
            }
            if (saleAnnouncementUpdateDto.Price <= 0)
            {
                return -3;
            }

            var saleAnnouncement = await
_estateAgencyDataStorage.SaleAnnouncements.GetAsync(saleAnnouncementUpdateDto.Id);
            saleAnnouncement.Price = saleAnnouncementUpdateDto.Price;
            saleAnnouncement.Status = "On review";

            _estateAgencyDataStorage.SaleAnnouncements.Update(saleAnnouncement);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return saleAnnouncement.Id;
        }

        public async Task<int> DeleteAsync(int id)
        {
            if (!await _estateAgencyDataStorage.SaleAnnouncements.ContainsEntityWithId(id))
            {
                return -5;
            }

            _estateAgencyDataStorage.SaleAnnouncements.Delete(id);
            await _estateAgencyDataStorage.SaveChangesAsync();
            return 1;
        }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace EstateAgency.Authentification.Services
{
    public interface IAuthentificationService
    {
        /// <summary>
        /// Logs user in.
        /// </summary>
        /// <param name="email">User email</param>
        /// <param name="password">User password</param>
        /// <returns>Returns success of operation</returns>
        Task<bool> LogIn(string email, string password);

        /// <summary>
        /// Logs user out.
        /// </summary>
        Task LogOut();
```

```
        /// <summary>
        /// Registers user.
        /// </summary>
        /// <param name="email">User email</param>
        /// <param name="password">User password</param>
        /// <param name="confirmPassword">User password for confirmation</param>
        /// <param name="roles">User roles</param>
        /// <returns>Returns collection of errors</returns>
        Task<IEnumerable<IdentityError>> Register(string email, string password, string
confirmPassword,
            params string[] roles);
    }
}

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace EstateAgency.Authentification.Services
{
    public class AuthentificationService : IAuthentificationService
    {
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;

        public AuthentificationService(SignInManager<User> signInManager, UserManager<User>
userManager, RoleManager<IdentityRole> roleManager)
        {
            _signInManager = signInManager;
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public async Task<bool> LogIn(string email, string password)
        {
            var result = await _signInManager.PasswordSignInAsync(email, password, false, false);
            return result.Succeeded;
        }

        public async Task LogOut()
        {
            await _signInManager.SignOutAsync();
        }

        public async Task<IEnumerable<IdentityError>> Register(string email, string password, string
confirmPassword, params string[] roles)
        {
            if (roles.Length == 0)
            {
```

```
            return new List<IdentityError> {new IdentityError {Description = "Amount of roles couldn't be 0."}};
        }

        var allRoles = _roleManager.Roles.ToList();
        var incorrectRoles = roles.Except(allRoles.Select(r => r.Name)).ToList();

        if (incorrectRoles.Any())
        {
            return new List<IdentityError>
            {
                new IdentityError {Description = $"Incorrect roles names: {string.Join(", ", incorrectRoles)}."}
            };
        }

        var user = new User { Email = email, UserName = email };

        var result = await _userManager.CreateAsync(user, password);
        if (!result.Succeeded)
        {
            return result.Errors;
        }

        await _userManager.AddToRolesAsync(user, roles);
        await _signInManager.SignInAsync(user, false);
        return new List<IdentityError>();
    }
  }
}

using Microsoft.AspNetCore.Identity;

namespace EstateAgency.Authentification
{
  public class User : IdentityUser
  {

  }
}

using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace EstateAgency.Authentification
{
  public class RoleInitializer
  {
    public static async Task InitializeAsync(UserManager<User> userManager, RoleManager<IdentityRole> roleManager)
    {
      var adminEmail = "admin@gmail.com";
```

```
            var adminPassword = "Qwerty123#";
            if (await roleManager.FindByNameAsync("admin") == null)
            {
                await roleManager.CreateAsync(new IdentityRole("admin"));
            }
            if (await roleManager.FindByNameAsync("user") == null)
            {
                await roleManager.CreateAsync(new IdentityRole("user"));
            }
            if (await userManager.FindByNameAsync(adminEmail) == null)
            {
                var adminUser = new User { Email = adminEmail, UserName = adminEmail };
                var result = await userManager.CreateAsync(adminUser, adminPassword);
                if (result.Succeeded)
                {
                    await userManager.AddToRoleAsync(adminUser, "admin");
                    await userManager.AddToRoleAsync(adminUser, "user");
                }
            }
        }
    }
}

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace EstateAgency.Authentification
{
    public class IdentityContext : IdentityDbContext<User>
    {
        public IdentityContext(DbContextOptions<IdentityContext> contextOptions) :
base(contextOptions)
        {
            Database.EnsureCreated();
        }
    }
}

namespace EstateAgency.API.Models.Announcements
{
    public class AnnouncementToReviewModel
    {
        public int Id { get; set; }
        public ApartmentInfoModel ApartmentInfo { get; set; }
        public ApartmentOwnerInfoModel ApartmentOwnerInfo { get; set; }
        public string Status { get; set; }
    }
}

namespace EstateAgency.API.Models.Announcements
{
    public class RentAnnouncementAddOrUpdateModel
```

```
    {
        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

using System;

namespace EstateAgency.API.Models.Announcements
{
    public class RentAnnouncementModel
    {
        public int Id { get; set; }
        public ApartmentInfoModel ApartmentInfo { get; set; }
        public ApartmentOwnerInfoModel ApartmentOwnerInfo { get; set; }
        public DateTime CreationDate { get; set; }
        public decimal Rent { get; set; }
        public int TermInDays { get; set; }
    }
}

namespace EstateAgency.API.Models.Announcements
{
    public class SaleAnnouncementAddOrUpdateModel
    {
        public int ApartmentId { get; set; }
        public int OwnerId { get; set; }

        public decimal Price { get; set; }
    }
}

using System;

namespace EstateAgency.API.Models.Announcements
{
    public class SaleAnnouncementModel
    {
        public int Id { get; set; }
        public ApartmentInfoModel ApartmentInfo { get; set; }
        public ApartmentOwnerInfoModel ApartmentOwnerInfo { get; set; }
        public DateTime CreationDate { get; set; }
        public decimal Price { get; set; }
    }
}

namespace EstateAgency.API.Models.ApartmentOwners
{
    public class ApartmentOwnerAddOrUpdateModel
```

```
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

namespace EstateAgency.API.Models.ApartmentOwners
{
    public class ApartmentOwnerModel
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}

namespace EstateAgency.API.Models.Apartments
{
    public class ApartmentAddOrUpdateModel
    {
        public int OwnerId { get; set; }

        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

namespace EstateAgency.API.Models.Apartments
{
    public class ApartmentModel
    {
        public int Id { get; set; }
        public ApartmentOwnerInfoModel ApartmentOwnerInfo { get; set; }
        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;

namespace EstateAgency.API.Models.Authentication
{
```

```
    public class UserLoginModel
    {
        [Required]
        public string Email { get; set; }

        [Required]
        public string Password { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;

namespace EstateAgency.API.Models.Authentication
{
    public class UserRegisterModel
    {
        [Required]
        public string Email { get; set; }

        [Required]
        public string Password { get; set; }

        [Required]
        public string PasswordConfirm { get; set; }

        [Required]
        public string[] Roles { get; set; }
    }
}

namespace EstateAgency.API.Models
{
    public class ApartmentInfoModel
    {
        public string Address { get; set; }
        public int NumberOfRooms { get; set; }
        public int Floor { get; set; }
        public double Area { get; set; }
        public string Description { get; set; }
    }
}

namespace EstateAgency.API.Models
{
    public class ApartmentOwnerInfoModel
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PhoneNumber { get; set; }
        public string Email { get; set; }
    }
}
```

```
using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.BLL.Announcements;

namespace EstateAgency.API.MappingConfigurations
{
    public class AnnouncementToReviewModelMappingProfile : Profile
    {
        public AnnouncementToReviewModelMappingProfile()
        {
            CreateMap<AnnouncementDto, AnnouncementToReviewModel>()
                .ForPath(dst => dst.ApartmentInfo.Description, src => src.MapFrom(rad =>
rad.ApartmentDescription))
                .ForPath(dst => dst.ApartmentInfo.Address, src => src.MapFrom(rad =>
rad.ApartmentAddress))
                .ForPath(dst => dst.ApartmentInfo.Area, src => src.MapFrom(rad => rad.ApartmentArea))
                .ForPath(dst => dst.ApartmentInfo.Floor, src => src.MapFrom(rad =>
rad.ApartmentFloor))
                .ForPath(dst => dst.ApartmentInfo.NumberOfRooms, src => src.MapFrom(rad =>
rad.ApartmentNumberOfRooms))
                .ForPath(dst => dst.ApartmentOwnerInfo.Email, src => src.MapFrom(rad =>
rad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwnerInfo.FirstName, src => src.MapFrom(rad =>
rad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwnerInfo.LastName, src => src.MapFrom(rad =>
rad.OwnerLastName))
                .ForPath(dst => dst.ApartmentOwnerInfo.PhoneNumber, src => src.MapFrom(rad =>
rad.OwnerPhoneNumber));
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.Apartments;
using EstateAgency.BLL.Apartments;

namespace EstateAgency.API.MappingConfigurations
{
    public class ApartmentDtoModelMappingProfile : Profile
    {
        public ApartmentDtoModelMappingProfile()
        {
            CreateMap<ApartmentDto, ApartmentModel>()
                .ForPath(dst => dst.ApartmentOwnerInfo.Email, src => src.MapFrom(ad =>
ad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwnerInfo.FirstName, src => src.MapFrom(ad =>
ad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwnerInfo.LastName, src => src.MapFrom(ad =>
ad.OwnerLastName))
                .ForPath(dst => dst.ApartmentOwnerInfo.PhoneNumber, src => src.MapFrom(ad =>
ad.OwnerPhoneNumber));
```

```
            CreateMap<ApartmentAddOrUpdateModel, ApartmentCreateDto>();
            CreateMap<ApartmentAddOrUpdateModel, ApartmentUpdateDto>();
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.ApartmentOwners;
using EstateAgency.BLL.ApartmentOwners;

namespace EstateAgency.API.MappingConfigurations
{
    public class ApartmentOwnerDtoModelMappingProfile : Profile
    {
        public ApartmentOwnerDtoModelMappingProfile()
        {
            CreateMap<ApartmentOwnerDto, ApartmentOwnerModel>();
            CreateMap<ApartmentOwnerAddOrUpdateModel, ApartmentOwnerCreateDto>();
            CreateMap<ApartmentOwnerAddOrUpdateModel, ApartmentOwnerUpdateDto>();
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.BLL.RentAnnouncements;

namespace EstateAgency.API.MappingConfigurations
{
    public class RentAnnouncementDtoModelMappingProfile : Profile
    {
        public RentAnnouncementDtoModelMappingProfile()
        {
            CreateMap<RentAnnouncementDto, RentAnnouncementModel>()
                .ForPath(dst => dst.ApartmentInfo.Description, src => src.MapFrom(rad =>
rad.ApartmentDescription))
                .ForPath(dst => dst.ApartmentInfo.Address, src => src.MapFrom(rad =>
rad.ApartmentAddress))
                .ForPath(dst => dst.ApartmentInfo.Area, src => src.MapFrom(rad => rad.ApartmentArea))
                .ForPath(dst => dst.ApartmentInfo.Floor, src => src.MapFrom(rad =>
rad.ApartmentFloor))
                .ForPath(dst => dst.ApartmentInfo.NumberOfRooms, src => src.MapFrom(rad =>
rad.ApartmentNumberOfRooms))
                .ForPath(dst => dst.ApartmentOwnerInfo.Email, src => src.MapFrom(rad =>
rad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwnerInfo.FirstName, src => src.MapFrom(rad =>
rad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwnerInfo.LastName, src => src.MapFrom(rad =>
rad.OwnerLastName))
                .ForPath(dst => dst.ApartmentOwnerInfo.PhoneNumber, src => src.MapFrom(rad =>
rad.OwnerPhoneNumber));
            CreateMap<RentAnnouncementAddOrUpdateModel, RentAnnouncementCreateDto>();
```

```
            CreateMap<RentAnnouncementAddOrUpdateModel, RentAnnouncementUpdateDto>();
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.BLL.SaleAnnouncements;

namespace EstateAgency.API.MappingConfigurations
{
    public class SaleAnnouncementDtoModelMappingProfile : Profile
    {
        public SaleAnnouncementDtoModelMappingProfile()
        {
            CreateMap<SaleAnnouncementDto, SaleAnnouncementModel>()
                .ForPath(dst => dst.ApartmentInfo.Description, src => src.MapFrom(sad =>
sad.ApartmentDescription))
                .ForPath(dst => dst.ApartmentInfo.Address, src => src.MapFrom(sad =>
sad.ApartmentAddress))
                .ForPath(dst => dst.ApartmentInfo.Area, src => src.MapFrom(sad => sad.ApartmentArea))
                .ForPath(dst => dst.ApartmentInfo.Floor, src => src.MapFrom(sad =>
sad.ApartmentFloor))
                .ForPath(dst => dst.ApartmentInfo.NumberOfRooms, src => src.MapFrom(sad =>
sad.ApartmentNumberOfRooms))
                .ForPath(dst => dst.ApartmentOwnerInfo.Email, src => src.MapFrom(sad =>
sad.OwnerEmail))
                .ForPath(dst => dst.ApartmentOwnerInfo.FirstName, src => src.MapFrom(sad =>
sad.OwnerFirstName))
                .ForPath(dst => dst.ApartmentOwnerInfo.LastName, src => src.MapFrom(sad =>
sad.OwnerLastName))
                .ForPath(dst => dst.ApartmentOwnerInfo.PhoneNumber, src => src.MapFrom(sad =>
sad.OwnerPhoneNumber));
            CreateMap<SaleAnnouncementAddOrUpdateModel, SaleAnnouncementCreateDto>();
            CreateMap<SaleAnnouncementAddOrUpdateModel, SaleAnnouncementUpdateDto>();
        }
    }
}

using EstateAgency.BLL.ApartmentOwners;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.ApartmentOwners
{
    public interface IApartmentOwnerResponseComposer
    {
        /// <summary>
        /// Composes response code for GetAll action in ApartmentOwnersController
        /// </summary>
        /// <param name="apartmentOwnerDtos">Apartment owner DTOs</param>
        /// <returns>Returns action result for GetAll action in ApartmentOwnersController</returns>
```

```
        IActionResult ComposeForGetAll(IEnumerable<ApartmentOwnerDto> apartmentOwnerDtos);

        /// <summary>
        /// Composes response code for Get action in ApartmentOwnersController
        /// </summary>
        /// <param name="apartmentOwnerDto">Apartment owner DTO</param>
        /// <returns>Returns action result for Get action in ApartmentOwnersController</returns>
        IActionResult ComposeForGet(ApartmentOwnerDto apartmentOwnerDto);

        /// <summary>
        /// Composes response code for Add action in ApartmentOwnersController
        /// </summary>
        /// <param name="statusCode">Code representing status of apartment owner addition
operation</param>
        /// <param name="apartmentOwnerCreateDto">Apartment owner DTO for creation</param>
        /// <returns>Returns action result for Add action in ApartmentOwnersController</returns>
        IActionResult ComposeForCreate(int statusCode, ApartmentOwnerCreateDto
apartmentOwnerCreateDto);

        /// <summary>
        /// Composes response code for Update action in ApartmentOwnersController
        /// </summary>
        /// <param name="statusCode">Code representing status of apartment owner updating
operation</param>
        /// <returns>Returns action result for Update action in ApartmentOwnersController</returns>
        IActionResult ComposeForUpdate(int statusCode);

        /// <summary>
        /// Composes response code for Delete action in ApartmentOwnersController
        /// </summary>
        /// <param name="statusCode">Code representing status of apartment owner deleting
operation</param>
        /// <returns>Returns action result for Delete action in ApartmentOwnersController</returns>
        IActionResult ComposeForDelete(int statusCode);
    }
}

using AutoMapper;
using EstateAgency.API.Models.ApartmentOwners;
using EstateAgency.BLL.ApartmentOwners;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.ApartmentOwners
{
    public class ApartmentOwnerResponseComposer : IApartmentOwnerResponseComposer
    {
        private readonly IMapper _mapper;

        public ApartmentOwnerResponseComposer(IMapper mapper)
        {
            _mapper = mapper;
```

```
    }

    public IActionResult ComposeForGetAll(IEnumerable<ApartmentOwnerDto>
apartmentOwnerDtos)
    {
        var apartmentOwnerModels =
_mapper.Map<IEnumerable<ApartmentOwnerModel>>(apartmentOwnerDtos);
        return new OkObjectResult(apartmentOwnerModels);
    }

    public IActionResult ComposeForGet(ApartmentOwnerDto apartmentOwnerDto)
    {
        if (apartmentOwnerDto == null)
        {
            return new NotFoundResult();
        }

        var apartmentOwnerModel = _mapper.Map<ApartmentOwnerModel>(apartmentOwnerDto);
        return new OkObjectResult(apartmentOwnerModel);
    }

    public IActionResult ComposeForCreate(int statusCode, ApartmentOwnerCreateDto
apartmentOwnerCreateDto)
    {
        switch (statusCode)
        {
            case -1:
                return new BadRequestObjectResult("Invalid first name.");
            case -2:
                return new BadRequestObjectResult("Invalid laste name.");
            case -3:
                return new BadRequestObjectResult("Invalid email.");
            case -4:
                return new BadRequestObjectResult("Invalid phone number.");
            default:
                return new CreatedAtRouteResult("GetApartmentOwner", new { Id = statusCode },
apartmentOwnerCreateDto);
        }
    }

    public IActionResult ComposeForUpdate(int statusCode)
    {
        switch (statusCode)
        {
            case -1:
                return new BadRequestObjectResult("Invalid first name.");
            case -2:
                return new BadRequestObjectResult("Invalid laste name.");
            case -3:
                return new BadRequestObjectResult("Invalid email.");
            case -4:
                return new BadRequestObjectResult("Invalid phone number.");
```

```
            default:
                return new OkResult();
        }
    }

    public IActionResult ComposeForDelete(int statusCode)
    {
        switch (statusCode)
        {
            case -5:
                return new NotFoundResult();
            default:
                return new NoContentResult();
        }
    }
  }
}

using EstateAgency.BLL.Apartments;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.Apartments
{
  public interface IApartmentResponseComposer
  {
    /// <summary>
    /// Composes response code for GetAll action in ApartmentsController
    /// </summary>
    /// <param name="apartmentDtos">Apartment DTOs</param>
    /// <returns>Returns action result for GetAll action in ApartmentsController</returns>
    IActionResult ComposeForGetAll(IEnumerable<ApartmentDto> apartmentDtos);

    /// <summary>
    /// Composes response code for Get action in ApartmentsController
    /// </summary>
    /// <param name="apartmentDto">Apartment DTO</param>
    /// <returns>Returns action result for Get action in ApartmentsController</returns>
    IActionResult ComposeForGet(ApartmentDto apartmentDto);

    /// <summary>
    /// Composes response code for Add action in ApartmentsController
    /// </summary>
    /// <param name="statusCode">Code representing status of apartment addition
operation</param>
    /// <param name="apartmentCreateDto">Apartment DTO for creation</param>
    /// <returns>Returns action result for Add action in ApartmentsController</returns>
    IActionResult ComposeForCreate(int statusCode, ApartmentCreateDto apartmentCreateDto);

    /// <summary>
    /// Composes response code for Update action in ApartmentsController
    /// </summary>
```

```
        /// <param name="statusCode">Code representing status of apartment updating
operation</param>
        /// <returns>Returns action result for Update action in ApartmentsController</returns>
        IActionResult ComposeForUpdate(int statusCode);

        /// <summary>
        /// Composes response code for Delete action in ApartmentsController
        /// </summary>
        /// <param name="statusCode">Code representing status of apartment deleting
operation</param>
        /// <returns>Returns action result for Delete action in ApartmentsController</returns>
        IActionResult ComposeForDelete(int statusCode);
    }
}

using AutoMapper;
using EstateAgency.API.Models.Apartments;
using EstateAgency.BLL.Apartments;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.Apartments
{
    public class ApartmentResponseComposer : IApartmentResponseComposer
    {
        private readonly IMapper _mapper;

        public ApartmentResponseComposer(IMapper mapper)
        {
            _mapper = mapper;
        }

        public IActionResult ComposeForGetAll(IEnumerable<ApartmentDto> apartmentDtos)
        {
            var apartmentModels = _mapper.Map<IEnumerable<ApartmentModel>>(apartmentDtos);
            return new OkObjectResult(apartmentModels);
        }

        public IActionResult ComposeForGet(ApartmentDto apartmentDto)
        {
            if (apartmentDto == null)
            {
                return new NotFoundResult();
            }

            var apartmentModel = _mapper.Map<ApartmentModel>(apartmentDto);
            return new OkObjectResult(apartmentModel);
        }

        public IActionResult ComposeForCreate(int statusCode, ApartmentCreateDto
apartmentCreateDto)
        {
```

```
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid number of rooms.");
                case -3:
                    return new BadRequestObjectResult("Invalid area.");
                case -4:
                    return new BadRequestObjectResult("Invalid floor.");
                case -5:
                    return new BadRequestObjectResult("Invalid adress.");
                default:
                    return new CreatedAtRouteResult("GetApartment", new { Id = statusCode },
apartmentCreateDto);
            }
        }

        public IActionResult ComposeForUpdate(int statusCode)
        {
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid number of rooms.");
                case -3:
                    return new BadRequestObjectResult("Invalid area.");
                case -4:
                    return new BadRequestObjectResult("Invalid floor.");
                case -5:
                    return new BadRequestObjectResult("Invalid adress.");
                default:
                    return new OkResult();
            }
        }

        public IActionResult ComposeForDelete(int statusCode)
        {
            switch (statusCode)
            {
                case -6:
                    return new NotFoundResult();
                default:
                    return new NoContentResult();
            }
        }
    }
}

using System.Collections.Generic;
using EstateAgency.BLL.RentAnnouncements;
```

```
using Microsoft.AspNetCore.Mvc;

namespace EstateAgency.API.Services.RentAnnouncements
{
    public interface IRentAnnouncementResponseComposer
    {
        /// <summary>
        /// Composes response code for GetAll action in RentAnnouncementsController
        /// </summary>
        /// <param name="rentAnnouncementDtos">Rent announcement DTOs</param>
        /// <returns>Returns action result for GetAll action in RentAnnouncementsController</returns>
        IActionResult ComposeForGetAll(IEnumerable<RentAnnouncementDto>
rentAnnouncementDtos);

        /// <summary>
        /// Composes response code for Get action in RentAnnouncementsController
        /// </summary>
        /// <param name="rentAnnouncementDto">Rent announcement DTO</param>
        /// <returns>Returns action result for Get action in RentAnnouncementsController</returns>
        IActionResult ComposeForGet(RentAnnouncementDto rentAnnouncementDto);

        /// <summary>
        /// Composes response code for Add action in RentAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of rent announcement addition
operation</param>
        /// <param name="rentAnnouncementCreateDto">Rent announcement DTO for
creation</param>
        /// <returns>Returns action result for Add action in RentAnnouncementsController</returns>
        IActionResult ComposeForCreate(int statusCode, RentAnnouncementCreateDto
rentAnnouncementCreateDto);

        /// <summary>
        /// Composes response code for Update action in RentAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of rent announcement updating
operation</param>
        /// <returns>Returns action result for Update action in RentAnnouncementsController</returns>
        IActionResult ComposeForUpdate(int statusCode);

        /// <summary>
        /// Composes response code for Delete action in RentAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of rent announcement deleting
operation</param>
        /// <returns>Returns action result for Delete action in RentAnnouncementsController</returns>
        IActionResult ComposeForDelete(int statusCode);
    }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
```

```
using EstateAgency.BLL.RentAnnouncements;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.RentAnnouncements
{
    public class RentAnnouncementResponseComposer : IRentAnnouncementResponseComposer
    {
        private readonly IMapper _mapper;

        public RentAnnouncementResponseComposer(IMapper mapper)
        {
            _mapper = mapper;
        }

        public IActionResult ComposeForGetAll(IEnumerable<RentAnnouncementDto>
rentAnnouncementDtos)
        {
            var rentAnnouncementModels =
_mapper.Map<IEnumerable<RentAnnouncementModel>>(rentAnnouncementDtos);
            return new OkObjectResult(rentAnnouncementModels);
        }

        public IActionResult ComposeForGet(RentAnnouncementDto rentAnnouncementDto)
        {
            if (rentAnnouncementDto == null)
            {
                return new NotFoundResult();
            }

            var rentAnnouncementModel =
_mapper.Map<RentAnnouncementModel>(rentAnnouncementDto);
            return new OkObjectResult(rentAnnouncementModel);
        }

        public IActionResult ComposeForCreate(int statusCode, RentAnnouncementCreateDto
rentAnnouncementCreateDto)
        {
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid apartment id.");
                case -3:
                    return new BadRequestObjectResult("Invalid rent value.");
                case -4:
                    return new BadRequestObjectResult("Invalid rent period value.");
                default:
                    return new CreatedAtRouteResult("GetRentAnnouncement", new { Id = statusCode },
rentAnnouncementCreateDto);
            }
```

```
        }

        public IActionResult ComposeForUpdate(int statusCode)
        {
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid apartment id.");
                case -3:
                    return new BadRequestObjectResult("Invalid rent value.");
                case -4:
                    return new BadRequestObjectResult("Invalid rent period value.");
                default:
                    return new OkResult();
            }
        }

        public IActionResult ComposeForDelete(int statusCode)
        {
            switch (statusCode)
            {
                case -5:
                    return new NotFoundResult();
                default:
                    return new NoContentResult();
            }
        }
    }
}

using EstateAgency.BLL.SaleAnnouncements;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.SaleAnnouncements
{
    public interface ISaleAnnouncementResponseComposer
    {
        /// <summary>
        /// Composes response code for GetAll action in SaleAnnouncementsController
        /// </summary>
        /// <param name="saleAnnouncementDtos">Sale announcement DTOs</param>
        /// <returns>Returns action result for GetAll action in SaleAnnouncementsController</returns>
        IActionResult ComposeForGetAll(IEnumerable<SaleAnnouncementDto>
saleAnnouncementDtos);

        /// <summary>
        /// Composes response code for Get action in SaleAnnouncementsController
        /// </summary>
        /// <param name="saleAnnouncementDto">Sale announcement DTO</param>
```

```
        /// <returns>Returns action result for Get action in SaleAnnouncementsController</returns>
        IActionResult ComposeForGet(SaleAnnouncementDto saleAnnouncementDto);

        /// <summary>
        /// Composes response code for Add action in SaleAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of sale announcement addition
operation</param>
        /// <param name="saleAnnouncementCreateDto">Sale announcement DTO for
creation</param>
        /// <returns>Returns action result for Add action in SaleAnnouncementsController</returns>
        IActionResult ComposeForCreate(int statusCode, SaleAnnouncementCreateDto
saleAnnouncementCreateDto);

        /// <summary>
        /// Composes response code for Update action in SaleAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of sale announcement updating
operation</param>
        /// <returns>Returns action result for Update action in SaleAnnouncementsController</returns>
        IActionResult ComposeForUpdate(int statusCode);

        /// <summary>
        /// Composes response code for Delete action in SaleAnnouncementsController
        /// </summary>
        /// <param name="statusCode">Code representing status of sale announcement deleting
operation</param>
        /// <returns>Returns action result for Delete action in SaleAnnouncementsController</returns>
        IActionResult ComposeForDelete(int statusCode);
    }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.BLL.SaleAnnouncements;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace EstateAgency.API.Services.SaleAnnouncements
{
    public class SaleAnnouncementResponseComposer : ISaleAnnouncementResponseComposer
    {
        private readonly IMapper _mapper;

        public SaleAnnouncementResponseComposer(IMapper mapper)
        {
            _mapper = mapper;
        }

        public IActionResult ComposeForGetAll(IEnumerable<SaleAnnouncementDto>
saleAnnouncementDtos)
        {
```

```
            var saleAnnouncementModels =
_mapper.Map<IEnumerable<SaleAnnouncementModel>>(saleAnnouncementDtos);
            return new OkObjectResult(saleAnnouncementModels);
        }

        public IActionResult ComposeForGet(SaleAnnouncementDto saleAnnouncementDto)
        {
            if (saleAnnouncementDto == null)
            {
                return new NotFoundResult();
            }

            var saleAnnouncementModel =
_mapper.Map<SaleAnnouncementModel>(saleAnnouncementDto);
            return new OkObjectResult(saleAnnouncementModel);
        }

        public IActionResult ComposeForCreate(int statusCode, SaleAnnouncementCreateDto
saleAnnouncementCreateDto)
        {
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid apartment id.");
                case -3:
                    return new BadRequestObjectResult("Invalid price value.");
                default:
                    return new CreatedAtRouteResult("GetSaleAnnouncement", new { Id = statusCode },
saleAnnouncementCreateDto);
            }
        }

        public IActionResult ComposeForUpdate(int statusCode)
        {
            switch (statusCode)
            {
                case -1:
                    return new BadRequestObjectResult("Invalid apartment owner id.");
                case -2:
                    return new BadRequestObjectResult("Invalid apartment id.");
                case -3:
                    return new BadRequestObjectResult("Invalid price value.");
                default:
                    return new OkResult();
            }
        }

        public IActionResult ComposeForDelete(int statusCode)
        {
            switch (statusCode)
```

```
            {
                case -5:
                    return new NotFoundResult();
                default:
                    return new NoContentResult();
            }
        }
    }
}

using EstateAgency.API.Models.Authentication;
using EstateAgency.Authentification.Services;
using Microsoft.AspNetCore.Mvc;
using System.Linq;
using System.Threading.Tasks;

namespace EstateAgency.API.Controllers
{
    [Route("api/[controller]")]
    public class AccountController : Controller
    {
        private readonly IAuthentificationService _authentificationService;

        public AccountController(IAuthentificationService authentificationService)
        {
            _authentificationService = authentificationService;
        }
        /// <summary>
        /// Performs user log in.
        /// </summary>
        /// <param name="model">User log in model</param>
        /// <response code="200">If the log in action succeeded</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [HttpPost("LogIn")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> LogIn([FromBody] UserLoginModel model)
        {
            var logInSuccess = await _authentificationService.LogIn(model.Email, model.Password);
            if (!logInSuccess)
            {
                return BadRequest();
            }
            return Ok();
        }

        /// <summary>
        /// Performs user log out.
        /// </summary>
        /// <response code="200">Always</response>
        [HttpPost("LogOut")]
        [ProducesResponseType(200)]
```

```
        public async Task<IActionResult> LogOut()
        {
            await _authentificationService.LogOut();
            return Ok();
        }

        /// <summary>
        /// Performs user registration.
        /// </summary>
        /// <param name="model">User registration model</param>
        /// <response code="200">If the registration action succeeded</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [HttpPost("Register")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> Register([FromBody] UserRegisterModel model)
        {
            var identityErrors =
                await _authentificationService.Register(model.Email, model.Password,
model.PasswordConfirm, model.Roles);
            if (identityErrors.Count() != 0)
            {
                return BadRequest(identityErrors);
            }
            return Ok();
        }
    }
}

using System.Collections.Generic;
using System.Threading.Tasks;
using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.BLL.Announcements.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EstateAgency.API.Controllers
{
    [Authorize(Roles = "admin")]
    [Route("api/[controller]")]
    public class AnnouncementsController : Controller
    {
        private readonly IAnnouncementService _announcementService;
        private readonly IMapper _mapper;

        public AnnouncementsController(IAnnouncementService announcementService, IMapper
mapper)
        {
            _announcementService = announcementService;
            _mapper = mapper;
        }
```

```
        /// <summary>
        /// Gets announcements with "On Review" status.
        /// </summary>
        /// <returns>Returns announcements with "On Review" status</returns>
        /// <response code="200">Always</response>
        [HttpGet]
        [ProducesResponseType(200)]
        public async Task<IActionResult> GetAllToReviewAsync()
        {
            var announcementToReviewDtos =  await _announcementService.GetAllToReviewAsync();
            return
Ok(_mapper.Map<IEnumerable<AnnouncementToReviewModel>>(announcementToReviewDtos));
        }

        /// <summary>
        /// Gets announcement by id.
        /// </summary>
        /// <param name="id">Announcement id</param>
        /// <returns>Returns announcement by id</returns>
        /// <response code="200">If the item exists</response>
        /// <response code="404">If the item is not found</response>
        [HttpGet("{id}")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetAsync(int id)
        {
            var announcementToReviewDto = await _announcementService.GetAsync(id);
            if (announcementToReviewDto == null)
            {
                return NotFound();
            }
            return Ok(_mapper.Map<AnnouncementToReviewModel>(announcementToReviewDto));
        }

        /// <summary>
        /// Updates announcement status.
        /// </summary>
        /// <param name="id">Announcement id</param>
        /// <param name="status">Announcement status</param>
        /// <response code="204">If the item updated</response>
        /// <response code="400">If the status is invalid</response>
        [HttpPut]
        [ProducesResponseType(204)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> UpdateStatusAsync(int? id, [FromBody] string status)
        {
            if (!id.HasValue || string.IsNullOrEmpty(status))
            {
                return BadRequest();
            }
            await _announcementService.UpdateStatusAsync(id.Value, status);
```

```
            return Ok();
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.ApartmentOwners;
using EstateAgency.API.Services.ApartmentOwners;
using EstateAgency.BLL.ApartmentOwners;
using EstateAgency.BLL.ApartmentOwners.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace EstateAgency.API.Controllers
{
    [Route("api/[controller]")]
    public class ApartmentOwnersController : Controller
    {
        private readonly IApartmentOwnerService _apartmentOwnerService;
        private readonly IMapper _mapper;
        private readonly IApartmentOwnerResponseComposer _apartmentOwnerResponseComposer;

        public ApartmentOwnersController(IApartmentOwnerService apartmentOwnerService,
IMapper mapper,
            IApartmentOwnerResponseComposer apartmentOwnerResponseComposer)
        {
            _apartmentOwnerService = apartmentOwnerService;
            _mapper = mapper;
            _apartmentOwnerResponseComposer = apartmentOwnerResponseComposer;
        }

        /// <summary>
        /// Gets all apartment owners.
        /// </summary>
        /// <returns>Returns all apartments</returns>
        /// <response code="200">Always</response>
        [Authorize(Roles = "user")]
        [HttpGet]
        [ProducesResponseType(200)]
        public async Task<IActionResult> GetAllAsync()
        {
            var apartmentOwnerDtos = await _apartmentOwnerService.GetAllAsync();
            var response =
_apartmentOwnerResponseComposer.ComposeForGetAll(apartmentOwnerDtos);
            return response;
        }

        /// <summary>
        /// Gets apartment owner by id.
        /// </summary>
        /// <param name="id">Apartment owner id</param>
```

```
        /// <returns>Returns apartment owner by id</returns>
        /// <response code="200">If the item exists</response>
        /// <response code="404">If the item is not found</response>
        [Authorize(Roles = "user")]
        [HttpGet("{id}", Name = "GetApartmentOwner")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetAsync(int id)
        {
            var apartmentOwnerDto = await _apartmentOwnerService.GetAsync(id);
            var response = _apartmentOwnerResponseComposer.ComposeForGet(apartmentOwnerDto);
            return response;
        }

        /// <summary>
        /// Creates apartment owner.
        /// </summary>
        /// <param name="apartmentOwnerAddOrUpdateModel">Apartment owner model</param>
        /// <returns>Returns route to created apartment owner</returns>
        /// <response code="201">If the item created</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "user")]
        [HttpPost]
        [ProducesResponseType(201)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> AddAsync([FromBody]
ApartmentOwnerAddOrUpdateModel apartmentOwnerAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var apartmentOwnerCreateDto =
_mapper.Map<ApartmentOwnerCreateDto>(apartmentOwnerAddOrUpdateModel);
            var statusCode = await _apartmentOwnerService.AddAsync(apartmentOwnerCreateDto);
            var response = _apartmentOwnerResponseComposer.ComposeForCreate(statusCode,
apartmentOwnerCreateDto);
            return response;
        }

        /// <summary>
        /// Updates apartment owner.
        /// </summary>
        /// <param name="id">Apartment owner id</param>
        /// <param name="apartmentOwnerAddOrUpdateModel">Apartment owner model</param>
        /// <response code="204">If the item updated</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "admin")]
        [HttpPut]
        [ProducesResponseType(204)]
        [ProducesResponseType(400)]
```

```
        public async Task<IActionResult> UpdateAsync(int? id, [FromBody]
ApartmentOwnerAddOrUpdateModel apartmentOwnerAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            if (id.HasValue)
            {
                var apartmentOwnerUpdateDto =
                    _mapper.Map<ApartmentOwnerUpdateDto>(apartmentOwnerAddOrUpdateModel);
                apartmentOwnerUpdateDto.Id = id.Value;
                var statusCode = await
_apartmentOwnerService.UpdateAsync(apartmentOwnerUpdateDto);
                var response = _apartmentOwnerResponseComposer.ComposeForUpdate(statusCode);
                return response;
            }
            else
            {
                var apartmentOwnerCreateDto =
_mapper.Map<ApartmentOwnerCreateDto>(apartmentOwnerAddOrUpdateModel);
                var statusCode = await _apartmentOwnerService.AddAsync(apartmentOwnerCreateDto);
                var response = _apartmentOwnerResponseComposer.ComposeForCreate(statusCode,
apartmentOwnerCreateDto);
                return response;
            }
        }

        /// <summary>
        /// Deletes apartment owner.
        /// </summary>
        /// <param name="id">Apartment owner id</param>
        /// <response code="204">If the item deleted</response>
        /// <response code="404">If the item not found</response>
        [Authorize(Roles = "admin")]
        [HttpDelete("{id}")]
        [ProducesResponseType(204)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Delete(int id)
        {
            var statusCode = await _apartmentOwnerService.DeleteAsync(id);
            var response = _apartmentOwnerResponseComposer.ComposeForDelete(statusCode);
            return response;
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.Apartments;
using EstateAgency.API.Services.Apartments;
using EstateAgency.BLL.Apartments;
```

```
using EstateAgency.BLL.Apartments.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace EstateAgency.API.Controllers
{
    [Route("api/[controller]")]
    public class ApartmentsController : Controller
    {
        private readonly IApartmentService _apartmentService;
        private readonly IMapper _mapper;
        private readonly IApartmentResponseComposer _apartmentResponseComposer;

        public ApartmentsController(IApartmentService apartmentService, IMapper mapper,
            IApartmentResponseComposer apartmentResponseComposer)
        {
            _apartmentService = apartmentService;
            _mapper = mapper;
            _apartmentResponseComposer = apartmentResponseComposer;
        }

        /// <summary>
        /// Gets all apartments.
        /// </summary>
        /// <returns>Returns all apartments</returns>
        /// <response code="200">Always</response>
        [Authorize(Roles = "user")]
        [HttpGet]
        [ProducesResponseType(200)]
        public async Task<IActionResult> GetAllAsync()
        {
            var apartmentDtos = await _apartmentService.GetAllAsync();
            var response = _apartmentResponseComposer.ComposeForGetAll(apartmentDtos);
            return response;
        }

        /// <summary>
        /// Gets apartment by id.
        /// </summary>
        /// <param name="id">Apartment id</param>
        /// <returns>Returns apartment by id</returns>
        /// <response code="200">If the item exists</response>
        /// <response code="404">If the item is not found</response>
        [Authorize(Roles = "user")]
        [HttpGet("{id}", Name = "GetApartment")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetAsync(int id)
        {
            var apartmentDto = await _apartmentService.GetAsync(id);
            var response = _apartmentResponseComposer.ComposeForGet(apartmentDto);
```

```
            return response;
        }

        /// <summary>
        /// Creates apartment.
        /// </summary>
        /// <param name="apartmentAddOrUpdateModel">Apartment model</param>
        /// <returns>Returns route to created apartment</returns>
        /// <response code="201">If the item created</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "user")]
        [HttpPost]
        [ProducesResponseType(201)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> AddAsync([FromBody] ApartmentAddOrUpdateModel
apartmentAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var apartmentCreateDto =
_mapper.Map<ApartmentCreateDto>(apartmentAddOrUpdateModel);
            var statusCode = await _apartmentService.AddAsync(apartmentCreateDto);
            var response = _apartmentResponseComposer.ComposeForCreate(statusCode,
apartmentCreateDto);
            return response;
        }

        /// <summary>
        /// Updates apartment.
        /// </summary>
        /// <param name="id">Apartment id</param>
        /// <param name="apartmentAddOrUpdateModel">Apartment model</param>
        /// <response code="204">If the item updated</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "admin")]
        [HttpPut]
        [ProducesResponseType(204)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> UpdateAsync(int? id, [FromBody]
ApartmentAddOrUpdateModel apartmentAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            if (id.HasValue)
            {
                var apartmentUpdateDto =
```

```
                _mapper.Map<ApartmentUpdateDto>(apartmentAddOrUpdateModel);
            apartmentUpdateDto.Id = id.Value;
            var statusCode = await _apartmentService.UpdateAsync(apartmentUpdateDto);
            var response = _apartmentResponseComposer.ComposeForUpdate(statusCode);
            return response;
        }
        else
        {
            var apartmentCreateDto =
_mapper.Map<ApartmentCreateDto>(apartmentAddOrUpdateModel);
            var statusCode = await _apartmentService.AddAsync(apartmentCreateDto);
            var response = _apartmentResponseComposer.ComposeForCreate(statusCode,
apartmentCreateDto);
            return response;
        }
    }

    /// <summary>
    /// Deletes apartment.
    /// </summary>
    /// <param name="id">Apartment id</param>
    /// <response code="204">If the item deleted</response>
    /// <response code="404">If the item not found</response>
    [Authorize(Roles = "admin")]
    [HttpDelete("{id}")]
    [ProducesResponseType(204)]
    [ProducesResponseType(404)]
    public async Task<IActionResult> Delete(int id)
    {
        var statusCode = await _apartmentService.DeleteAsync(id);
        var response = _apartmentResponseComposer.ComposeForDelete(statusCode);
        return response;
    }
  }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.API.Services.RentAnnouncements;
using EstateAgency.BLL.RentAnnouncements;
using EstateAgency.BLL.RentAnnouncements.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace EstateAgency.API.Controllers
{
  [Route("api/[controller]")]
  public class RentAnnouncementsController : Controller
  {
    private readonly IRentAnnouncementService _rentAnnouncementService;
    private readonly IMapper _mapper;
```

```
        private readonly IRentAnnouncementResponseComposer
_rentAnnouncementResponseComposer;
        private readonly IRentAnnouncementExpressionComposer
_rentAnnouncementExpressionComposer;

        public RentAnnouncementsController(IRentAnnouncementService rentAnnouncementService,
IMapper mapper,
            IRentAnnouncementResponseComposer rentAnnouncementResponseComposer,
IRentAnnouncementExpressionComposer rentAnnouncementExpressionComposer)
        {
            _rentAnnouncementService = rentAnnouncementService;
            _mapper = mapper;
            _rentAnnouncementResponseComposer = rentAnnouncementResponseComposer;
            _rentAnnouncementExpressionComposer = rentAnnouncementExpressionComposer;
        }

        /// <summary>
        /// Gets rent announcement by id.
        /// </summary>
        /// <param name="id">Rent announcement id</param>
        /// <returns>Returns rent announcement by id</returns>
        /// <response code="200">If the item exists</response>
        /// <response code="404">If the item is not found</response>
        [HttpGet("{id}", Name = "GetRentAnnouncement")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetAsync(int id)
        {
            var rentAnnouncementDto = await _rentAnnouncementService.GetAsync(id);
            var response =
_rentAnnouncementResponseComposer.ComposeForGet(rentAnnouncementDto);
            return response;
        }

        /// <summary>
        /// Finds rent announcements by criteria.
        /// </summary>
        /// <param name="maxPrice">Max price in rent announcement</param>
        /// <param name="numberOfRooms">Number of rooms in rent announcement</param>
        /// <param name="adress">Adress in rent announcement</param>
        /// <returns>Returns rent announcement by criteria</returns>
        /// <response code="200">Always</response>
        [HttpGet]
        [ProducesResponseType(200)]
        public async Task<IActionResult> FindAsync(int? maxPrice, int? numberOfRooms, string
adress)
        {
            var expression = _rentAnnouncementExpressionComposer.Compose(maxPrice,
numberOfRooms, adress);
            var rentAnnouncementDtos = await _rentAnnouncementService.FindAsync(expression);
            var response =
_rentAnnouncementResponseComposer.ComposeForGetAll(rentAnnouncementDtos);
```

```
            return response;
        }

        /// <summary>
        /// Creates rent announcement.
        /// </summary>
        /// <param name="rentAnnouncementAddOrUpdateModel">Rent announcement
model</param>
        /// <returns>Returns route to created rent announcement</returns>
        /// <response code="201">If the item created</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "user")]
        [HttpPost]
        [ProducesResponseType(201)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> AddAsync([FromBody]
RentAnnouncementAddOrUpdateModel rentAnnouncementAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var rentAnnouncementCreateDto =
_mapper.Map<RentAnnouncementCreateDto>(rentAnnouncementAddOrUpdateModel);
            var statusCode = await
_rentAnnouncementService.AddAsync(rentAnnouncementCreateDto);
            var response = _rentAnnouncementResponseComposer.ComposeForCreate(statusCode,
rentAnnouncementCreateDto);
            return response;
        }

        /// <summary>
        /// Updates rent announcement.
        /// </summary>
        /// <param name="id">Rent announcement id</param>
        /// <param name="rentAnnouncementAddOrUpdateModel">Rent announcement
model</param>
        /// <response code="204">If the item updated</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "admin")]
        [HttpPut]
        [ProducesResponseType(204)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> UpdateAsync(int? id, [FromBody]
RentAnnouncementAddOrUpdateModel rentAnnouncementAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }
```

```
            if (id.HasValue)
            {
                var rentAnnouncementUpdateDto =

_mapper.Map<RentAnnouncementUpdateDto>(rentAnnouncementAddOrUpdateModel);
                rentAnnouncementUpdateDto.Id = id.Value;
                var statusCode = await
_rentAnnouncementService.UpdateAsync(rentAnnouncementUpdateDto);
                var response = _rentAnnouncementResponseComposer.ComposeForUpdate(statusCode);
                return response;
            }
            else
            {
                var rentAnnouncementCreateDto =
_mapper.Map<RentAnnouncementCreateDto>(rentAnnouncementAddOrUpdateModel);
                var statusCode = await
_rentAnnouncementService.AddAsync(rentAnnouncementCreateDto);
                var response = _rentAnnouncementResponseComposer.ComposeForCreate(statusCode,
rentAnnouncementCreateDto);
                return response;
            }
        }

        /// <summary>
        /// Deletes rent announcement.
        /// </summary>
        /// <param name="id">Rent announcement id</param>
        /// <response code="204">If the item deleted</response>
        /// <response code="404">If the item not found</response>
        [Authorize(Roles = "admin")]
        [HttpDelete("{id}")]
        [ProducesResponseType(204)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Delete(int id)
        {
            var statusCode = await _rentAnnouncementService.DeleteAsync(id);
            var response = _rentAnnouncementResponseComposer.ComposeForDelete(statusCode);
            return response;
        }
    }
}

using AutoMapper;
using EstateAgency.API.Models.Announcements;
using EstateAgency.API.Services.SaleAnnouncements;
using EstateAgency.BLL.SaleAnnouncements;
using EstateAgency.BLL.SaleAnnouncements.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace EstateAgency.API.Controllers
```

```
{
    [Route("api/[controller]")]
    public class SaleAnnouncementsController : Controller
    {
        private readonly ISaleAnnouncementService _saleAnnouncementService;
        private readonly IMapper _mapper;
        private readonly ISaleAnnouncementResponseComposer
_saleAnnouncementResponseComposer;
        private readonly ISaleAnnouncementExpressionComposer
_saleAnnouncementExpressionComposer;

        public SaleAnnouncementsController(ISaleAnnouncementService saleAnnouncementService,
IMapper mapper,
            ISaleAnnouncementResponseComposer saleAnnouncementResponseComposer,
            ISaleAnnouncementExpressionComposer saleAnnouncementExpressionComposer)
        {
            _saleAnnouncementService = saleAnnouncementService;
            _mapper = mapper;
            _saleAnnouncementResponseComposer = saleAnnouncementResponseComposer;
            _saleAnnouncementExpressionComposer = saleAnnouncementExpressionComposer;
        }

        /// <summary>
        /// Gets sale announcement by id.
        /// </summary>
        /// <param name="id">Sale announcement id</param>
        /// <returns>Returns sale announcement by id</returns>
        /// <response code="200">If the item exists</response>
        /// <response code="404">If the item is not found</response>
        [HttpGet("{id}", Name = "GetSaleAnnouncement")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> GetAsync(int id)
        {
            var saleAnnouncementDto = await _saleAnnouncementService.GetAsync(id);
            var response =
_saleAnnouncementResponseComposer.ComposeForGet(saleAnnouncementDto);
            return response;
        }

        /// <summary>
        /// Finds sale announcements by criteria.
        /// </summary>
        /// <param name="maxPrice">Max price in sale announcement</param>
        /// <param name="numberOfRooms">Number of rooms in sale announcement</param>
        /// <param name="adress">Adress in sale announcement</param>
        /// <returns>Returns sale announcement by criteria</returns>
        /// <response code="200">Always</response>
        [HttpGet]
        [ProducesResponseType(200)]
        public async Task<IActionResult> FindAsync(int? maxPrice, int? numberOfRooms, string
adress)
```

```
        {
            var expression = _saleAnnouncementExpressionComposer.Compose(maxPrice,
numberOfRooms, adress);
            var saleAnnouncementDtos = await _saleAnnouncementService.FindAsync(expression);
            var response =
_saleAnnouncementResponseComposer.ComposeForGetAll(saleAnnouncementDtos);
            return response;
        }

        /// <summary>
        /// Creates sale announcement.
        /// </summary>
        /// <param name="saleAnnouncementAddOrUpdateModel">Sale announcement
model</param>
        /// <returns>Returns route to created sale announcement</returns>
        /// <response code="201">If the item created</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "user")]
        [HttpPost]
        [ProducesResponseType(201)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> AddAsync([FromBody]
SaleAnnouncementAddOrUpdateModel saleAnnouncementAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var saleAnnouncementCreateDto =
_mapper.Map<SaleAnnouncementCreateDto>(saleAnnouncementAddOrUpdateModel);
            var statusCode = await
_saleAnnouncementService.AddAsync(saleAnnouncementCreateDto);
            var response = _saleAnnouncementResponseComposer.ComposeForCreate(statusCode,
saleAnnouncementCreateDto);
            return response;
        }

        /// <summary>
        /// Updates sale announcement.
        /// </summary>
        /// <param name="id">Sale announcement id</param>
        /// <param name="saleAnnouncementAddOrUpdateModel">Sale announcement
model</param>
        /// <response code="204">If the item updated</response>
        /// <response code="400">If the model is invalid or contains invalid data</response>
        [Authorize(Roles = "admin")]
        [HttpPut]
        [ProducesResponseType(204)]
        [ProducesResponseType(400)]
        public async Task<IActionResult> UpdateAsync(int? id, [FromBody]
SaleAnnouncementAddOrUpdateModel saleAnnouncementAddOrUpdateModel)
```

```
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            if (id.HasValue)
            {
                var saleAnnouncementUpdateDto =

_mapper.Map<SaleAnnouncementUpdateDto>(saleAnnouncementAddOrUpdateModel);
                saleAnnouncementUpdateDto.Id = id.Value;
                var statusCode = await
_saleAnnouncementService.UpdateAsync(saleAnnouncementUpdateDto);
                var response = _saleAnnouncementResponseComposer.ComposeForUpdate(statusCode);
                return response;
            }
            else
            {
                var saleAnnouncementCreateDto =
_mapper.Map<SaleAnnouncementCreateDto>(saleAnnouncementAddOrUpdateModel);
                var statusCode = await
_saleAnnouncementService.AddAsync(saleAnnouncementCreateDto);
                var response = _saleAnnouncementResponseComposer.ComposeForCreate(statusCode,
saleAnnouncementCreateDto);
                return response;
            }
        }

        /// <summary>
        /// Deletes sale announcement.
        /// </summary>
        /// <param name="id">Sale announcement id</param>
        /// <response code="204">If the item deleted</response>
        /// <response code="404">If the item not found</response>
        [Authorize(Roles = "admin")]
        [HttpDelete("{id}")]
        [ProducesResponseType(204)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Delete(int id)
        {
            var statusCode = await _saleAnnouncementService.DeleteAsync(id);
            var response = _saleAnnouncementResponseComposer.ComposeForDelete(statusCode);
            return response;
        }
    }
}

{
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
```

```
      "LogLevel": {
        "Default": "Warning"
      }
    },
    "Console": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  },
  "ConnectionStrings": {
    "EstateAgency": "Server=LAPTOP-
JD40D77L\\FORWOTCH;Database=EstateAgencyDb;Trusted_Connection=True;",
    "EstateAgencyAuthentification": "Server=LAPTOP-
JD40D77L\\FORWOTCH;Database=EstateAgencyAuthentificationDb;Trusted_Connection=True;"
  }
}

using EstateAgency.API.Services.ApartmentOwners;
using EstateAgency.API.Services.Apartments;
using EstateAgency.API.Services.RentAnnouncements;
using EstateAgency.API.Services.SaleAnnouncements;
using EstateAgency.Authentification;
using EstateAgency.Authentification.Services;
using EstateAgency.BLL.Announcements.Services;
using EstateAgency.BLL.ApartmentOwners.Services;
using EstateAgency.BLL.Apartments.Services;
using EstateAgency.BLL.RentAnnouncements.Services;
using EstateAgency.BLL.SaleAnnouncements.Services;
using EstateAgency.DAL.EF;
using EstateAgency.DAL.UnitOfWork;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.OpenApi.Models;

namespace EstateAgency.API
{
    public static class DependencyInjectionExtensions
    {
        public static IServiceCollection ResolveDalDependencies(this IServiceCollection services,
string connectionString)
        {
            services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(connectionString));
            services.AddScoped<IUnitOfWork, UnitOfWork>();
            return services;
        }

        public static IServiceCollection ResolveServicesDependencies(this IServiceCollection services)
        {
```

```
            services.AddScoped<IRentAnnouncementService, RentAnnouncementService>();
            services.AddScoped<IRentAnnouncementResponseComposer,
RentAnnouncementResponseComposer>();
            services.AddScoped<IRentAnnouncementExpressionComposer,
RentAnnouncementExpressionComposer>();
            services.AddScoped<IRentAnnouncementCreator, RentAnnouncementCreator>();

            services.AddScoped<ISaleAnnouncementService, SaleAnnouncementService>();
            services.AddScoped<ISaleAnnouncementResponseComposer,
SaleAnnouncementResponseComposer>();
            services.AddScoped<ISaleAnnouncementExpressionComposer,
SaleAnnouncementExpressionComposer>();
            services.AddScoped<ISaleAnnouncementCreator, SaleAnnouncementCreator>();

            services.AddScoped<IApartmentService, ApartmentService>();
            services.AddScoped<IApartmentResponseComposer, ApartmentResponseComposer>();
            services.AddScoped<IApartmentCreator, ApartmentCreator>();

            services.AddScoped<IApartmentOwnerService, ApartmentOwnerService>();
            services.AddScoped<IApartmentOwnerResponseComposer,
ApartmentOwnerResponseComposer>();

            services.AddScoped<IAnnouncementService, AnnouncementService>();

            services.AddScoped<IAuthentificationService, AuthentificationService>();

            return services;
        }

        public static IServiceCollection ResolveIdentityDependencies(this IServiceCollection services,
            string connectionString)
        {
            services.AddDbContext<IdentityContext>(options =>
                options.UseSqlServer(connectionString));

            services.AddIdentity<User, IdentityRole>()
                .AddEntityFrameworkStores<IdentityContext>();

            services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
                .AddCookie(options =>
                {
                    options.LoginPath = new Microsoft.AspNetCore.Http.PathString("/Account/Login");
                });

            return services;
        }

        public static IServiceCollection RegisterSwagger(this IServiceCollection services)
        {
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo
```

```
            {
                Title = "Estate Agency API",
                Version = "v1"
            });
            c.IncludeXmlComments(
                @"bin\Debug\netcoreapp2.0\EstateAgency.API.xml");
        });

        return services;
    }
  }
}

using AutoMapper;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Newtonsoft.Json;

namespace EstateAgency.API
{
  public class Startup
  {
    public Startup(IConfiguration configuration)
    {
      Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
      services.AddMvc().AddJsonOptions(option =>
        option.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore);

      services.ResolveDalDependencies(Configuration.GetConnectionString("EstateAgency"));
      services.ResolveServicesDependencies();

services.ResolveIdentityDependencies(Configuration.GetConnectionString("EstateAgencyAuthentification"));
      services.RegisterSwagger();

      services.AddAutoMapper();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
      if (env.IsDevelopment())
```

```
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseAuthentication();

            app.UseSwagger();
            app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "Estate Agency API v1"));

            app.UseMvc();
        }
    }
}

using EstateAgency.Authentification;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using System;
using System.Threading.Tasks;

namespace EstateAgency.API
{
    public class Program
    {
        public async static Task Main(string[] args)
        {
            var host = BuildWebHost(args);

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;
                try
                {
                    var userManager = services.GetRequiredService<UserManager<User>>();
                    var rolesManager = services.GetRequiredService<RoleManager<IdentityRole>>();
                    await RoleInitializer.InitializeAsync(userManager, rolesManager);
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred while seeding the database.");
                }
            }

            host.Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
```

```
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
        }
    }
```